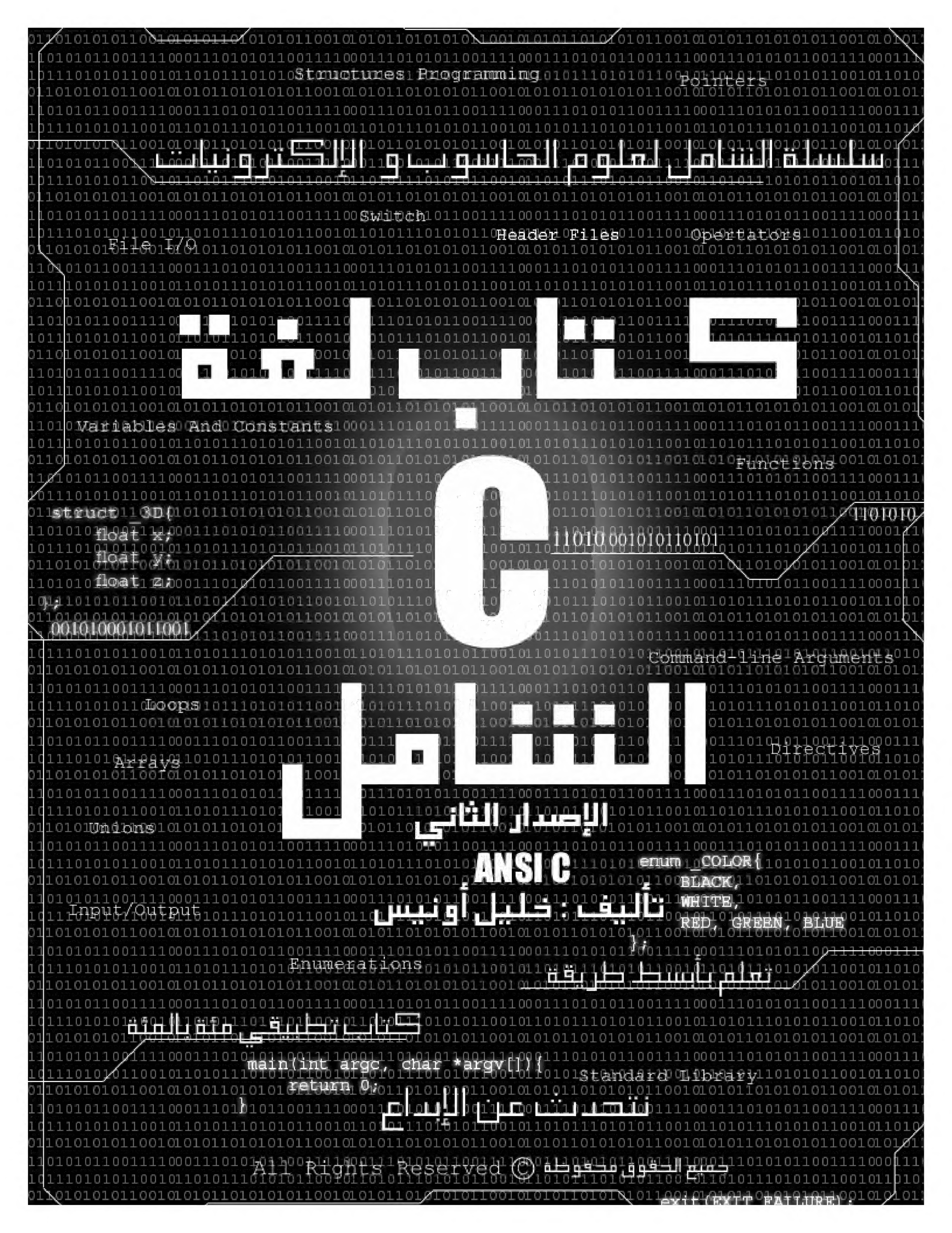

Structures Programming
Pointers
سلسلة الشامل لعلوم الحاسوب و الإلكترونيات
Switch
File I/O
Header Files
Opertators
كتاب لغة
Variables And Constants
Functions
C
struct _3D{
float x;
float y;
float z;
};
Command-line Arguments
Loops
الشامل
Directives
Arrays
الإصدار الثاني
Unions
ANSI C
enum _COLOR{
BLACK,
WHITE,
RED, GREEN, BLUE
};
تأليف : خليل أونيس
Input/Output
Enumerations
تعلم بأبسط طريقة
كتاب تطبيقي مئة بالمئة
main(int argc, char *argv[]){
return 0;
}
Standard Library
نتحدث عن الإبداع
All Rights Reserved © جميع الحقوق محفوظة



محتويات الكتاب بنظرة سريعة

جدول المحتويات

حول الكتاب

الحمد **الله** رب العالمين و الصلاة و السلام على سيد المرسلين نبينا **محمد** *صلى الله عليه* و على آله و صحبه أجمعين...أما بعد، إقتربت المدة عام من إنشاء النسخة الأولى من كتاب لغة *C* الشامل، و اليوم قمت بإنشاء النسخة الثانية منه، و لكن هذه النسخة لم أركز على التوسيع فيها على اسابقة، إنما ركزت على تصحيح أخطاءها، صححت الذي استطعت ملاحظته، و لاَزِلْتُ أنتظر ملاحظة أخطاء أخرى من قُراء هذه الكتاب، لذا أرجو لكل من قرأ الكتاب و وجد به أخطاء سواء كانت أخطاء إملائية أو معنوية (و خاصة المعنوية) أن يقوم بتنبيهي على بريدي الإليكتروني khalil_ounis@yahoo.com، أو يمكن أن يوضع الخطأ في موضوع هذا الكتاب. حيث سيتم إنشاء قائمة لأسماء مُلاحظي أخطاء هذا الكتاب و وضْعها في الصفحات الأولى من الكتاب (يعني عمل جماعي).

و من أخطاء النسخة الأولى(هذا لكي أبين لقراء النسخة الأولى من هذا الكتاب أي يمكنهم إعادة القراءة) من هذا الكتاب كانت في كل من: الفصل الأول، بعض الأخطاء في الجزء "البدء مع لغة *C*"، و بعض الأماكن التي تجاهلتها في قسم "المؤثرات *Operators*" و قسم "المتغيرات و الثوابت"، أما الفصل الثاني فتوجد أخطاء في جزء "المؤشرات"، و أخيرا الفصل الثالث في كل من "*Command-line Arguments*" و "المكتبة القياسية". و طبعا توجد أخطاء أخرى متفرقة في الكتاب لم أذكرها. و أكيد هناك إضافات في الكتاب(لاكن لا تتوقع الكثير).

فصول هذه النسخة مثل النسخة السابقة:

- في الفصل الأول مفاهيم و مبادئ أساسية في لغة *C*: الإدخال و الإخراج، التعليقات و المؤثرات، القرارات و عناصر لغة *C*، مع ملخص للفصل الأول.
- الفصل الثاني مكمل للفصل الأول في كل من القرار *Switch*، حلقات التكرار، المصفوفات و المؤشرات، الدوال، الملفات الرأسية، الإدخال و الإخراج في الملفات و التراكيب، و أخيرا ملخص للفصل الثاني.
- الفصل الثالث مكمل للكتاب، مع إضافة أهم ثوابت، مختصرات و دوال المكتبة القياسية للغة *C*.

و الكتاب مفتوح لكل من يريد إضافة حرف، كلمة، جملة أو جُمل، أجزاء أو أي شيء مفيد(شاركنا الخير). أي تعليقات أو ملاحظات أهلا و سهلا.

خليل أونيس، الجزائر

٠٦٤٥٧٦٦١٨ (من الداخل)، ٢١٣٦٤٥٧٦٦١٨+

تاريخ الإنتهاء من النسخة: ١٩-٠٨-٢٠٠٦

المقدمة

في أيام بداية الحاسوب كانت البرمجة تتم على لغة بدائية منخفضة المستوى *low-level language* تدعى بلغة الآلة *Machine language*، حيث كانت تفهمها الآلة مباشرة، و يتم البرمجة عليها بأوامر تُمثل بخيوط طويلة مُكونة من الواحد و الصفر (الصفر تعني *low* و هي محصورة بين -٠,٥ و +٠,٥ فولت، و الواحد يعني *high* و هو محصور بين +٤,٥ و +٥,٥ فولت) أي بما يسمى بالنظام الثنائي، و كانت البرمجة عليها صعبة و معقدة حتى تم تطوير لغة التجميع *assembly language*، و هي من اللغات المنخفضة المستوى *low-level languages* أيضا، حيث كانت سهلة بالنسبة للغة الآلة، فبدل استعمال سلاسل من الصفر و الواحد نستعمل أوامر ذات كلمات مفهومة مثل *ADD* و *MOV*.

مع مرور الوقت تم تطوير لغات برمجة أخرى مثل *COBOL*، *BASIC* و *C*، و كان التعامل معها بالكلمات و النصوص مما جعل هذه اللغات مقروئة. و هذه معلومات مختصرة عن بعض اللغات:

لغة الجميع *Assembly Language*، تم تطويرها في عام ١٩٥٦ من قبل شركة *IBM*. لغة فورتران *Fortran Language*، و كلمة *Fortran* مختصرة من *Formula Translation* أي صيغة الترجمة، تم تطويرها في عام ١٩٥٤ من قبل فريق يترأسه جون باكوس *John Backus*، حيث كانت تستخدم بكثرة في التطبيقات الرياضية. لغة كوبول *COBOL Language*، إختصار لـ *Common Business Oriented Language*، أي لغة موجهة للأعمال التجارية، تم تطويرها في عام ١٩٥٩ من قبل لجنة قصيرة مكونة من ثلاثة شركات منها *IBM*. لغة البازيك *Basic Language*، إختصار لـ *Beginner's All-Purpose Symbolic Instruction Code*، أي شفرة رموز لتعليمات جميع الأغراض للمبتدأين، و تم تطويرها في عام ١٩٦٣ من قبل الأستاذ جون كيمني *John Kemeny* و الأستاذ توماس كورز *Thomas Kurtz*، و أخذت شهرة كبيرة (إلى حد الآن). لغة آلقول *Algol Language*، إختصار لـ *Algorithmic Language*، تم تطويرها في عام ١٩٥٨ من قبل مجموعة من علماء حواسيب أوروبيين و أمريكيين. لغة باسكال *Pascal Language*، و اسم هذه اللغة مأخوذ من اسم العالم الفرنسي *Blaise Pascal*، و تم تطويرها عام ١٩٧٠ من قبل نيكلوس ويرث *Niklaus Wirth*. و تذكر أنا كل من اللغات السابقة هي أقدم لغات برمجة.

لغة *C* من لغات الأغراض العامة، و تستعمل بكثرة في برمجة النظم *Systems Programming* و أنظمة التشغيل *Operating Systems*، تم تطويرها في السبعينات من طرف كين تومسن *Ken Thompson* و دنيس ريتشي *Dennis Ritchie* في مختبرات *Bell*. لغة *C* من اللغات المنخفضة المستوى *low-level languages* حيث أنها قريبة من الأجهزة و شبيها بلغة التجميع *assembly language* في عملها، و لكن البعض يعتبرها لغة متوسطة المستوى *mid-level language* لأنها لغة تحاكي لغة الإنسان بعض الشيء، و هي لغة مستقلة عن البنية الصلبة للحاسوب.

قام كين تومسن و دنيس ريتشي بتطوير لغة *C* لبرمجة نظام يونيكس *Unix*، حيث ركزا مطوري هذه اللغة على أن تكون لغتهم سهلة الاستعمال حيث يمكن كتابة برامج كبيرة مع قلة الأخطاء و في وقت أقصر. في عام ١٩٧٣ تم إطلاق لغة *C* بشكل رسمي، و سميت بلغة *C* لأنها كانت مشتقة من لغة الـ *B* (و كانت لغة الـ *B* نفسها مشتقة من لغة *BCPL* التي قام بتطويرها مارتن ريتشاردز *Martin Richards* في عام ١٩٦٧، و هي مختصرة من *Basic Combined Programming Language*، حيث كان الفرق بين اللغتين هو نوع البيانات) التي قام بتطويرها كين تومسن في عام ١٩٦٩ حيث أخذ الحرف *B* من اسم المختبر *Bell* الذي يعمل به، و الذي يلي الحرف *B* في الأبجدية هو *C*، و ذلك هو سبب تسميتها بلغة *C*. في عام ١٩٧٨ قام دنيس ريتشي و براين كارنيغان *Brian Kernighan* بتأليف أول كتاب لهذه اللغة و سمي بـ *The C Programming Language* و الذي يعتبر المرجع الأساسي لهذه اللغة، و كان الكتاب معروف بنسخة *K&R C* (*Kernighan & Ritchie C*) و السبب في تسميته بـ *K&R C* هو كثرة استعمال لغة *C* بشكل كبير و الذي أدى إلى تطوير مكتبات و دوال في نسخ مختلفة من لغة *C* حتى أصبح كل من تلك النسخ غير متوافقة مع بعضها و كادت أن تكون غير متشابها، و هذا ما أدى إلى تعريف نسخة قياسية للغة *C*. في عام ١٩٨٩ تم إطلاق النسخة القياسية للغة *C* و سميت بـ *ANSI C* و هي مختصرة من *American National Standards Institute C* أي اللجنة الوطنية الأميركية للمعايير، و بتعاون بين اللجنة الوطنية الأميركية للمعايير و المنظمة العالمية للمعايير تم إطلاق لغة *C* القياسية في مختلف أنحاء العالم و سميت بـ *ISO C* و هي إختصار لـ *International Organization for Standardization*. و كانت النسخة القياسية للغة *C* مختلفة بعض الشيء عن نسخة *K&R C* (في عام ١٩٨٨ قام دنيس ريتشي و براين كارنيغان بكتابة النسخة الثانية من كتاب *The C Programming Language* لنسخة القياسية للغة *C*، أي *ANSI C*).



دنيس ريتشي　　　　كين تومسن

الفصل الأول – أساسيات في لغة C

1

مقدمة: في هذا الفصل سنتعلم المبادئ الأولية على كيفية البرمجة في لغة C، كيفية الإعلان عن المتغيرات و الثوابـــت، الإدخال و الإخراج، مع الجمل الشرطية.

بالتوفيق إن شاء الله

قال الله تعالى:

﴿ يرفع الله الذين آمنوا منكم والذين أوتوا العلم درجات ﴾

صدق الله تعالى

١,١ الأدوات اللازمة

أدوات لغة *C* هي ثلاثة أشياء لا أكثر، محرر نصوص *texts editor*، مترجم *compiler* و مربط *linker*. و أي برنامج تتم كتابته يجب أن يمر على هذه الأدوات، و في نهاية ينتج الملف التنفيذي. و لا يمكن الاستغناء عن أداة من هذه الأدوات. في المترجمات الحديثة أصبح كل من الأدوات مدمجة مع بعضها مما جعلها أكثر سهول في الاستعمال، فمثلا لو أردنا ترجمة و ربط برنامج، زر واحد من لوحة المفاتيح أو نقرة من الفأرة تقوم ترجمة و ربط المشروع ثم تنفيذ البرنامج. أما في السابقة فكانت تتم هذه العمليات على شكل أوامر من *Console*.

١,١,١ محرر نصوص **texts editor**:

الخطوة الأول في البرمجة هي كتابة البرنامج، و طبعا ذلك يتم عبر محررات نصوص، و في لغة *C* نقوم بكتابة البرامج على أي محرر نصوص، فقط نراعي أن يتم حفظ مصدر البرنامج على صيغة *c.*، هناك بعض المترجمات (القديمة) التي لا يهمها امتداد الملف النصي للبرنامج، و لكن من الأفضل استعمال الصيغة الرسمية. و من شروط الملفات النصية للغة *C* أن تكون النصوص مكتوبة بنظام *ASCII*، مثلا محرر *KWrite* في أنظمة *Linux* و *Notepad* في أنظمة *Windows*، كِلا من محررين يعتمدان على شفرة *ASCII*. لا يمكن استعمال المحرر *Word* في أنظمة *Windows* أو *KWord* في أنظمة *Linux*.

١,١,٢ مترجم **compiler**:

تقوم المترجمات بترجمة أو تحويل الملفات المصدرية إلى لغة منخفضة المستوى إن لم تكون هناك أخطاء في قواعد اللغة، يمكن أن تترجم إلى لغة التجميع *Assembly Language* أو إلى لغة الآلة *Machine Language* مباشرة، حيث بعد الترجمة يتم إنشاء ملفات بصيغة *obj*.. تحتوي هذه الملفات على تعليمات التجميع أو الآلة مم يسهل عملية ربط لغتين أو أكثر مع بعضها، فمثلا يمكننا استدعاء دوال من لغة *Pascal* في لغة *C*.

يوجد العديد من المترجمات في أغلب الأنظمة، مثلا في أنظمة *Windows* يوجد المترجم *Visual C++* حيث يقوم بترجمة كلا اللغتين *C* و *C++*، و هو مقدم من طرف شركة *MicroSoft*، و يوجد كذلك المترجم *Dev-C++* و المقدم من شركة *Bloodshed*، و مترجمات أخرى مثل *Turbo C*، *Quick C*، *Pelles C*، .... بالنسبة للمترجم *Visual C++* فهو غير مجاني. المترجم *Dev-C++* من المترجمات المجانية و يمكن تحميله من الرابط التالي:

http://www.bloodshed.net/devcpp.html

المترجم *Turbo C* أيضا من المترجمات المجانية، و هو من أقدمها، و الأكثر استعمالنا في الجامعات، حيث يمكن تحميله من الرابط التالي:

http://www.pitt.edu/~stephenp/misc/downloadTC.html

المترجم *Pelles C* أيضا من المترجمات المجانية و يعتبر من أفضلها و يمكن تحميله من الرابط التالي:

http://www.smorgasbordet.com/pellesc/download.htm

أما في أنظمة *Unix* و *Linux*، فلا تحتاج إلى مترجمات لأنها مدمجة مع أي نسخة من نسخ *Unix* و *Linux*، كل ما تحتاجه هو محرر نصوص. و هذا لا يعني أنه لا يوجد مترجمات لتلك الأنظمة، بل يوجد و ربما عددها أكثر من التي هي موجودة على نظام *Windows*.

أدخل على الرابط التالي حيث توجد مترجمات مجانية عديدة في كل من أنظمة *Windows* و أنظمة *Linux*:

http://www.thefreecountry.com/compilers/cpp.shtml

جميع المترجمات الحديثة متوفرة بها *IDE*، ماذا يعني هذا المصطلح؟، أولا الكلمة *IDE* مختصرة من *Integrated Development Environment* أي بيئة تطوير متكاملة، حيث تساعد المترجمات ذات بيئة تطوير متكاملة على المبرمج في كل من التحرير، الترجمة و الربط، ففي السابق كانت الترجمة و الربط تتم على شكل أوامر، أما في البيئات التطوير المتكاملة فأصبحت عملية الربط و الترجمة تتم عبر زر واحد من لوحة المفاتيح أو عبر نقرة من الفأرة، أما كتابة البرامج فتتم عبر محررات نصوص مستقلة عن المترجم، و هذه الظاهرة موجودة إلى حد الآن في أنظمة *Unix/Linux*، حتى أداة *Qt* تعتمد على أوامر لترجمة و الربط. و هنا أيضا كنت أريد أن نرى كيف تتم الترجمة في هذا الأنظمة، و نأخذ مثالا لذلك، مثلا لدينا برنامج محفوظ بإسم *cprog.c*، فمن خط الأوامر (مثلا *Konsole*) نكتب:

```
cc cprog.c
```

هذه في حالة أن البرنامج معتمد على لغة *C* فقط، أما إذا كان مدمج مع لغة *C++* فسنكتب الأمر *gcc* بدل *cc*. و طبعا توجد طرق أخرى مثل:

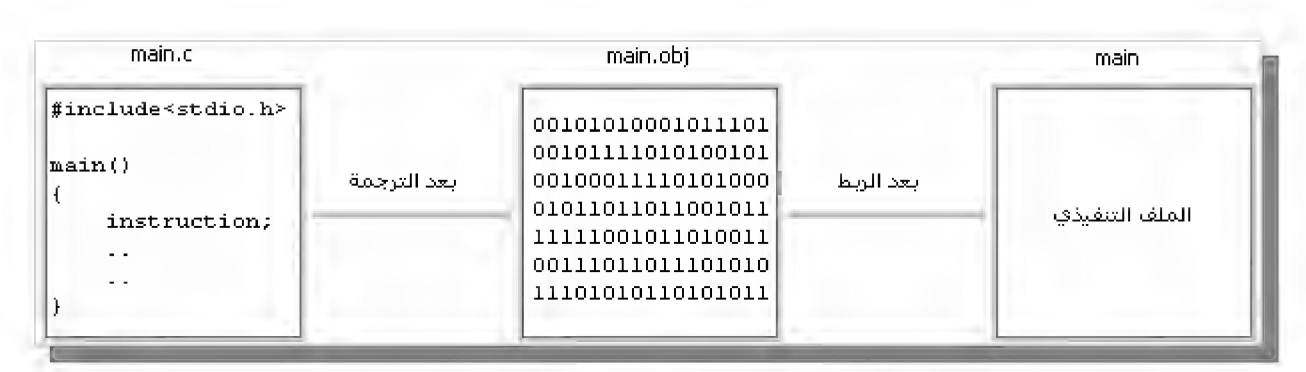
```
cc -o cprog cprog.c
```

هنا ستكون لديك إمكانية إعادة تسمية البرنامج، أما عملية تنفيذ البرنامج فتكون كتالي:

```
./cprog
```

١,١,٣ المربط *linker*:

يقوم المربط بجمع الملفات ذات الصيغة *obj.* ثم يعطينا البرامج التنفيذية و التي تكون غالبا بامتداد *exe.*، أو ملفـــات مكتبات الربط الديناميكية و التي تكون بإمتداد *dll.*، و يمكن أن تكون هذه الملفات مكتوبة بمختلف اللغات.



الشكل ١,١,١: مرحلة إنشاء ملف تنفيذي

١,٢ البدء مع لغة C

قم بتحميل و تثبيت أحد المترجمات السابقة و قم بتشغيلها كأي برنامج، ثم قم بإنشاء مشروع جديد للغة *C* في بيئة الــ *Console* مع إنشاء ملف نصي جديد و الحرص على أن يتم حفظه بامتداد *c.*، يمكن كتابــة *main.c* كإســم للملف النصي و الذي سنقوم بالكتابة عليه البرنامج الأول و هو:

```
1 #include<stdio.h>
2
3 main()
4 {
5       printf("Hello, World!");
6 }
```

البرنامج ١,٢,١: البرنامج الأول في لغة C

ملاحظة:

قم بكتابة البرنامج بدون الترقيمات، أي نكتب البرنامج على الشكل التالي:

```
#include<stdio.h>

main()
{
      printf("Hello, World!");
}
```

البرنامج ١,٢,١: البرنامج الأول في لغة C

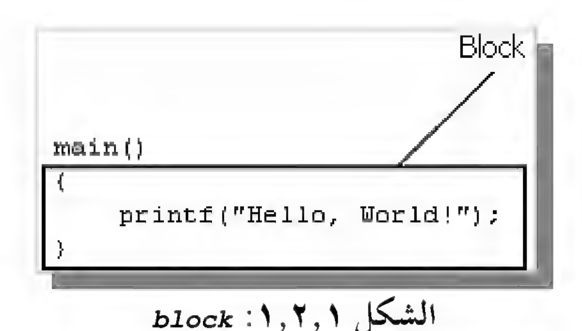
هذا من أبسط البرنامج التي يمكن كتابتها، يقوم هذا البرنامج عند ترجمته و تنفيذه بطباعة الجملــة *Hello, World* علــى الشاشة في بيئة الــ *Console*. السطر الأول من البرنامج به الشفرة *(Code)* `#include<stdio.h>` و هي مقسمة إلى قسمين، الأول هو `#include<>`، و غالبا ما تكون الكلمة `#include` أزرقة اللون.

و القسم الثاني هو ما بين الرمزين أكبر من و أصغر من < >، حيث يوجد الملف `stdio.h`، في هذا القسم نقوم بكتابة أسماء لملفات تسمى بالملفات الرأسية (يمكنك أن تراها في المجلد *include* من المترجم الذي تستعمله)، و هي عديدة و كل ملف منها له مجاله الخاص، حيث يحتوي على ثوابت و دوال تسهل علينا البرمجة. الملف الرأسي `stdio.h` مختصر مــن *Standard Input Output*، أما `.h` فهو امتداد الملف الرأسي و هو مختصر من *Header File*. فائدة الكلمــة `#include` هو ضم الملف الرأسي الموجود بين الرمزين أكبر من و أصغر من < > إلى مشروعنا. يوجد العديد من الملفات الرأسية، سنتطرق إليها فيما بعد.

في السطر الثالث يوجد اسم دالة و هي `main()` و هي الدالة الرئيسية لأي مشروع و لا يمكن الاستغناء عنـــها، و لا يمكن التغير في اسمها إلا في حالات. و من هذه الدالة يبدأ البرنامج بالتنفيذ بشكل مترتب، أما القوسين بعد اسم الدالـــة فهما اللذان يبينان على أنها دالة (و أنها دالة بدون وسائط) و ليست متغير أو ثابت. في السطر الرابع توجد الحاضنة `{` و التي تعني بداية الدالة `main`.

في السطر الخامس توجد الكلمة `printf` و هي عبارة عن دالة موجودة في الملف الرأسي `stdio.h`، و هي مختصرة من *print format*، أي صيغة الطبع، و هي تقوم بطبع (إخراج) ما هو بداخل أقواس الدالة إلى الشاشة، و في مثالنـــا هـــذا يوجد النص *Hello, World !* و هي الجملة التي سيتم إخراجها إلى الشاشة، و تكون الجمل دائما داخل اقتباسيين `" "`، و في نهاية السطر نكتب الفاصلة المنقوطة و هي تعني نهاية السطر التعليمة (أو التعليمات). تستعمل الدالة `printf` بصفة عامة في عرض أو إخراج معلومات إلى أداة الإخراج و هي الشاشة *Screen* الخاصة بالحاسوب.

و أخيرا السطر السادس حيث موجود به الحاضنة `}` و التي تعني نهاية الدالة الرئيسية `main`. و تسمى حاضنة البداية `{` و حاضنة النهاية `}` و ما بينهما بالـــ *block*، صورة توضيحية:



الشكل ١,٢,١: block

يمكن كتابة البرامج السابق بطرق مختلفة، حيث يمكن تقسيم الجملة *Hello, World !* إلى قسمين مثل:

```
1 #include<stdio.h>
2
3 main()
4 {
5       printf("Hello, ");
6       printf("World!");
7 }
```

البرنامج ١,٢,٢: البرنامج الأول في لغة C (٢)

و هنا سيتم طبع الجملة كاملة في سطر واحد، و تقسيمها لا يعني أن كل كلمة في سطر. و يمكن أيضا كتابـــة الجملـــة حرفيا، كل حرف بدالة من `printf`. أو يمكن كتابة الدالين في سطر واحد، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("Hello, "), printf("World!");
6  }
```

البرنامج ١,٢,٣: البرنامج الأول في لغة C (٣)

أو:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("Hello, "); printf("World!");
6  }
```

البرنامج ١,٢,٤: البرنامج الأول في لغة C (٤)

و توجد طريقة لا يمكن استعمالها و هي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("Hello,
6              World!");
7  }
```

البرنامج ١,٢,٥: البرنامج الأول في لغة C (٥)

عند ترجمة هذا المثال سينبهك المترجم عن وجود أخطاء، منها نسيان قوس النهاية لدالة `printf`، و لتفادي هذه الأخطاء نقوم بوضع *anti-slash* في نهاية السطر الأول من الدالة `printf`، و تصبح الدالة كالآتي:

```
     printf("Hello, \
           World!");
```

البرنامج ١,٢,٦: البرنامج الأول في لغة C (٦)

في المثال السابق إن كتبنا السطر الأول (الذي يتمثل في ضم الملف الرأسي `stdio.h`) في نهاية البرنامج فإن المترجم لن يجد الدالة `printf`، و ستنجم أخطاء عن ذلك، لذا يجب دائما أن يكون ضم الملفات الرأسية قبل الدوال المراد استعمالها و يستحسن دائما أن يتم ضم الملفات في بداية كل مشروع.

يمكن كتابة الكلمة *Hello,* في سطر و الكلمة *World !* في سطر آخر و ذلك بإضافة الرمز `\n` بين الكلمتين، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
```

```
5	printf("Hello, \nWorld!");
6 }
```

البرنامج ١,٢,٧: البرنامج الأول في لغة C (٧)

أو كتابة كل من الكلمات في دالة مثل:

```
1 #include<stdio.h>
2
3 main()
4 {
5	printf("Hello, ");
6	printf("\n");
7	printf("World!");
8 }
```

البرنامج ١,٢,٨: البرنامج الأول في لغة C (٨)

عند ترجمة البرنامج و تنفيذه فلن تجد الرمز `\n` و ستجد أن كل من الكلمتين في سطر، يتم استبدال الرمز `\n` بسطر جديد حيث لا يتم طباعة الرمز، و الحرف *n* يعني *New line*.

يمكن كتابة المثال الأول في ثلاثة أسطر كما في يلي:

```
1 #include<stdio.h>
2
3 main(){printf("Hello, World!");}
```

البرنامج ١,٢,٩: البرنامج الأول في لغة C (٩)

تم جمع جميع الأسطر في السطر الثالث، و البرنامج يعمل مثل السابق بدون أخطاء، حيث ستلاحظ أنه يمكن استعمال الحاضنة { (بداية الدالة) و الحاضنة } (نهاية الدالة) في نفس السطر، و يمكن استعمال أكثر من ذلك مثل:

```
1 #include<stdio.h>
2
3 main(){
4	printf("Hello, World!");}
```

البرنامج ١,٢,١٠: البرنامج الأول في لغة C (١٠)

و طرق أخرى، و لكن يجب أن تكون الأوامر، الوظائف و الدوال المراد إستعمالها داخل الحاضنتين { } لدالة الرئيسية. و مثل هذه الطرق لا يفضل استعمالها و خاصة إذا كان البرنامج كبير. و توجد طريقة لا يمكن إستعمالها و هي موضحة في المثال التالي:

```
1 #include<stdio.h> main(){
2	printf("Hello, World!");}
```

البرنامج ١,٢,١١: البرنامج الأول في لغة C (١١)

إذا ترجمة هذا المثال فسينبهك المترجم عن وجود خطأ لأن الكلمة `#include` تتطلب سطرا كاملا لها (من قواعد اللغة). تدعى الكلمة `#include` بالتوجيه *directive* أو قبل المعالج *preprocessor* و سميت بقبل المعالج لأنه يتم تنفيذها قبل الترجمة، و هي تقوم بضم محتويات الملف الرأسي المطلوب إلى المشروع، حيث يحتوي ذلك الملف الرأسي على مجموعة من ثوابت، بنيات و دوال تساعدنا في برمجة برامجنا. توجد الكثير من التوجيهات *directive* و يمكن تميزها بالرمز #، سنعرفها في الدروس القادمة.

يمكن أيضا وضع *block* داخل الدالة الرئيسية `main`، مثال:

```
1   #include<stdio.h>
2
3   main()
4   {
5       printf("Hello, World!\n");
6       {
7           printf("Hello, World!\n");
8       }
9       printf("Hello, World!\n");
10  }
```

البرنامج ١,٢,١٢: البرنامج الأول في لغة C (١٢)

و يتم التعامل معها كالتعامل مع *block* الدالة الرئيسية، و يمكن إنشاء أكثر من *block* داخل الدالة الرئيسية، أو استعمال *block* داخل *block* آخر.

١,٢,١ التعامل مع الأعداد:

التعامل مع الأعداد هو طباعة الأعداد على الشاشة و استعمال مؤثرات عليها مثل الجمع، الطرح، القسمة و الضرب، و سنتعامل مع الأعداد باستخدام الدالة `printf`، و ربما تقول أن الأمر سهل فقط نقوم بكتابة الأعداد التي نريدها داخل الاقتباسات في الدالة `printf`، صحيح يمكن استعمال تلك الطريقة و لكن المترجم هنا سيتعامل مع الأعداد على أنها نص ثابت و ليست أعداد، هذه الحالة لا يمكن استعمال عمليات رياضية عليها.

في لغة *C* لكل نوع من الأعداد رمز لتعامل معه، مثلا الأعداد الصحيحة يتم التعامل معها بالاستعمال الرمز `%d` أو `%i`، الرمز الأول مختصر من *Decimal* و الرمز الثاني مختصر من *Integer*، هذا بالنسبة للأعداد الصحيح، أما الأعداد الحقيقية فيتم التعامل معها باستخدام الرمز `%f`، و الحرف *f* مختصر من *float*، و أيضا توجد رموز أخرى خاصة بالتعامل مع كل من الحروف و النصوص، سنتطرق إليها فيما بعد.

نذهب إلى التعامل مع الأعداد الصحيحة، لكتابة عدد من نوع الأعداد الصحيحة نكتب كما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          printf("%d", 20);
6  }
```

البرنامج ١,٢,١٣:طباعة عدد صحيح

هنا وضعنا رمز الأعداد الصحيحة داخل الدالة printf و بين الاقتباسيين، و بعد الاقتباسيين نقوم بكتابة العدد المـــراد طبعه، و الحرص على أن يكون بين الاقتباسيين و العدد فاصلة، و بهذه الطريقة يمكن استعمال عمليات مثل الجمع مثلا و ذلك بإضافة مؤثر الجمع مع العدد المراد الجمع معه مثل ما هو موضح في المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          printf("%d", 20+5);
6  }
```

البرنامج ١,٢,١٤: استعمال الجمع

و يمكن استعمال باقي المؤثرات مثل الطرح، القسمة و الضرب بنفس الطريقة. و يمكن إظهار أكثر من رقـــم و ذلـــك بزيادة الرمز %d مع فصله من الرمز السابق حتى تكون الأرقام واضحة مثل ما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          printf("%d %d", 20+5, 87);
6  }
```

البرنامج ١,٢,١٥: طبع عددين

كلما نظيف رمز الأعداد الصحيحة نقوم بكتابة الرقم الإضافي بعد الرقم السابق و نفصلهما بفاصلة، يمكـــن اســـتعمال أكثر من عددين و بطريقة منظمة فمثلا إذا أردنا أن نقوم بكتابة عملية الجمع فسنكتب كما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          printf("%d + %d = %d\n", 20, 5, 20+5);
6  }
```

البرنامج ١,٢,١٦: عملية جمع

و نفس الطريقة مع الأعداد الحقيقية فقط نستعمل الرمز f% بدل الرمز d%. و هذا مثال لكيفية استعمالها:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("%f + %f = %f\n", 1.0, 2.14, 1.0+2.14);
6  }
```

البرنامج ١,٢,١٧: جمع و إظهار أعداد حقيقية

ملاحظة:

في المثال السابقة وضعنا النقطة في مكان الفاصلة، هكذا كي يميز المترجم على أنها قيم للأعداد حقيقية أي أنها أعداد لها فواصل، أما إذا وضعت الفاصلة في مكان النقطة فسيعتبرها المترجم منفصلة عن الأخرى و هكذا ستنجم أخطاء كثيرة، و تذكر أن الفاصلة تستعمل في فصل وسائط دالة.

أما رموز طبع الأحرف و النصوص فطريقة استعمالها مثل الطرق السابقة فقط بدل الرمزين d% و f% نضع c% للأحرف، حيث حرف *c* مختصر من *character* و نضع الرمز s% للنصوص، و الحرف *s* مختصر من *String* أي سلسلة حروف. بالنسبة للحروف فهذا مثال يوضح طريقة استعمالها:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("%c", "a");
6  }
```

البرنامج ١,٢,١٨: طباعة حرف

و يمكن استعمال هذا المثال أيضا:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("%c", 'a');
6  }
```

البرنامج ١,٢,١٩: طباعة حرف (٢)

هنا سيطبع البرنامج الحرف *a*، و في حالة أردنا طبع نص نكتب كما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
```

```
5 	printf("%s\n", "Hello, World!");
6 }
```

البرنامج ١,٢,٢٠: طباعة نص

و يمكن أيضا كتابة كل كلمة أو حرف في إقتباسين مثل:

```
1 #include<stdio.h>
2 
3 main()
4 {
5 	printf("%s", "Hello, " "\n" "World");
6 }
```

البرنامج ١,٢,٢١: طباعة نص (٢)

و توجد رموز أخرى منها من هي خاصة بأرقام النظام السداسي عشر و التي يكون التعامل معها باستخدام الرمز `%x` أو `%X` حيث تبدأ من `0x` أو `0X`، مثلا الرقم `0x000F`، و الرمز `%o` لأعداد النظام الثماني، و رموز أخرى سنعرفها في الدروس القادمة.

١,٢,٢ الأخطاء المحتملة:

١. في لغة C المترجمات تفرق بين الحروف الكبيرة و الحروف الصغيرة، مثلا الدالة `main` لا يمكن كتابتها `Main` أو `MAIN`.

٢. لا يمكن استعمال الدالة `printf` أو دوال أخرى خارج الدالة الرئيسية `main`، مثلا لا يمكن كتابة:

```
1 #include<stdio.h>
2 
3 printf("Hello, World!\n");
4 
5 main()
6 {
7 
8 }
```

البرنامج ١,٢,٢٢: الخطأ ١

إلا في حالة استعمال دوال بها دوال أخرى ثم ربطتها بالدالة الرئيسية، سنتعرف على ذلك في الدروس القادمة.

٣. كثيرا ما يتم نسيان الفاصلة المنقوطة، و إن تم نسيانها لا يمكن إنشاء الملف التنفيذي للبرنامج حتى يتم تصحيح الخطأ.

٤. لا يمكن استعمال الفاصلة المنقوطة في نهاية سطر الدالة الرئيسية `main()`، و سبب ذلك هو عندما يتم الإعلان عن دالة و إعطاءها أوامر لا يجب أن نكتب الفاصلة المنقوطة، ليست مثل دوال معرفة سابقا مثل الدالة `printf`.

٥. في دالة الطبع `printf`، إن كتبنا النص المراد طبعه بدون رموز الاقتباس " " فإن المترجم سينبهك عن وجود خطأ.

١,٢,٣ تمارين:

١. أكتب برنامج يقوم بطباعة الجملة *Hello, World !* مرتين، الأولى في سطر و الثانية في سطر آخر.

٢. أكتب برنامج يقوم بطباعة الجملة *Hello, World !*، كل حرف في سطر.

٣. هل يمكن تغير اسم الدالة الرئيسية `main` إلى اسم من اختيارنا؟

٤. هل يمكن الاستغناء عن الدالة الرئيسية `main`؟

٥. هل يمكن استعمال الدالة الرئيسية أكثر من مرة؟

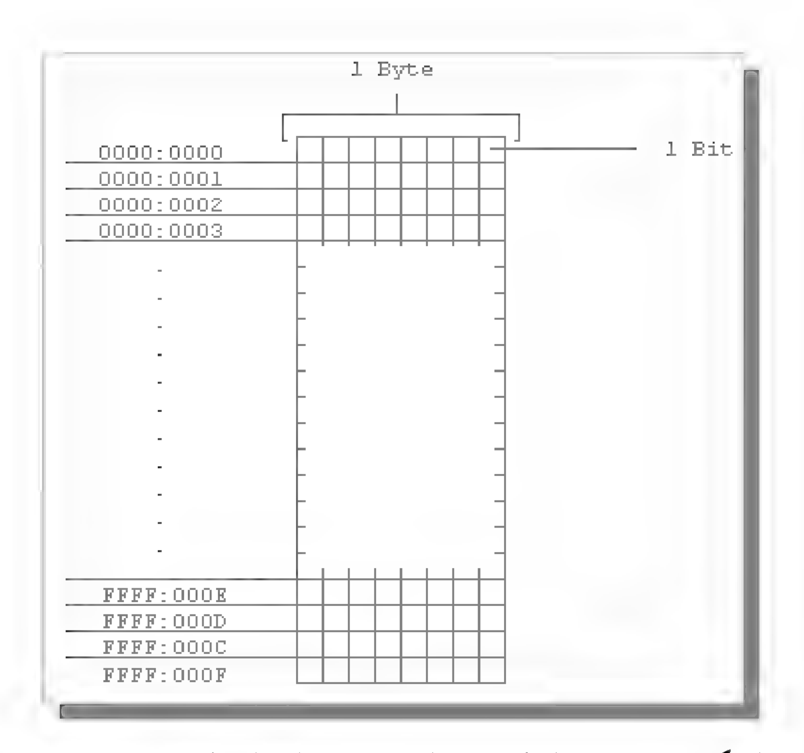
٦. أكتب برنامج يقوم بطبع نتيجة طرح العدد ٢ من ٣,٥.

٧. أكتب برنامج يقوم بطباعة الكلمة *Hello* باستخدام رموز الأحرف (`%c`).

٨. أكتب برنامج يقوم بكتابة نصيين باستعمال الرمز الخاص بطبع النصوص مرتين (`%s%s`).

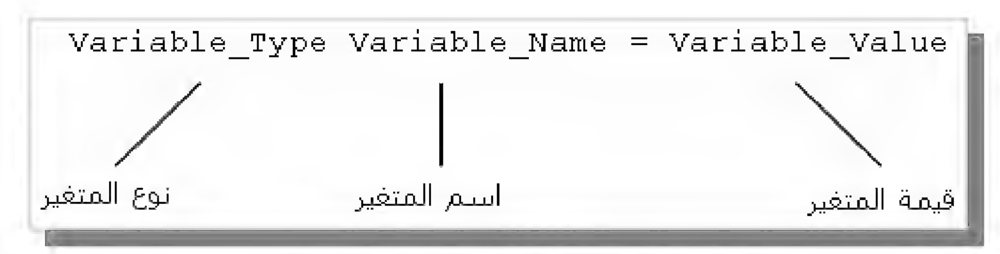
١,٣ المتغيرات و الثوابت Variables and Constants

المتغيرات هي مجموعة من البايتات يتم حجزها في الذاكرة العشوائية *Random Access Memory* أي *RAM*، حيث يتم وضع قِيم متغيرة في تلك البايتات المحجوزة، يمكن استرجاعها في أي وقت. كل بايت يتم الوصول إليها عبر عنوان. و تفقد هذه الذاكرة جميع بياناتها عند قطع التيار. الـــ *RAM* عبارة عن رقاقة *Chip* تحتوي على عدد من الترانزستورات *Transistors* و المكثفات *Capacitor* تقدر بالملايين، حيث واحد ترانزستور و مكثف يشكلان وحـــدة ذاكـــرة تقـــدر بالـــبت *Bit*، يحل هذا البت إما القيمة ٠ أو القيمة ١، حيث ٨ بتات تشكل واحد بايت *Byte*، و كل بيات من هـــذه الذاكرة يمثل بعنوان(يتم التعامل مع هذه العناوين بالنظام السداسي العشر) يمكن الوصول إليه و التغير في محتواه، مما نفهم أن الذاكرة عبارة عن عناوين متسلسلة، لكل عنوان قيمة متغيرة. صورة توضيحية:



الشكل ١,٣,١: الذاكرة و العناوين في النمط الحقيقي real mode

في لغة *C* يوجد عدت أنواع من المتغيرات و الثوابت، منها متغيرات خاص بالأعداد الصحيح و أخرى بالأعداد الحقيقية و أخرى بالأحرف و ...، و دائما نقوم بالإعلان عن المتغيرات و الثوابت قبل استعمالها(و كأنك تحجز مكانـــا أولا ثم تقوم بالجلوس (أي وضع قيمة)). طريقة الإعلان عن متغير هي كتابة نوع المتغير ثم اسم المتغير ثم القيمة التي ســـيحتويها (هذا في حالة إعطاءه قيمة مباشرة) المتغير، صورة توضيحية:



الشكل ١,٣,٢: طريقة الإعلان عن متغير

١,٣,١ نوع المتغير Variable Type:

كما قلنا سابقا، توجد عدة أنواع للمتغيرات، و لكن الذي يجب أن نعرفه هو أن تلك الأنواع لا تختلف عن بعضها إلا في الحجم، و هذا يعني لو أعلنا عن متغير لأعداد صحيح يمكننا أن نعطيه حرفا بدل من قيمة صحيح، و عكس. و توجد حالة خاصة هنا و هي الأعداد الحقيقية لأنها ليست كغيرها. أنواع المتغيرات هي:

١,٣,١,١ متغير الأعداد الصحيحة int:

نقوم بالإعلان عن متغير من نوع الأعداد الصحيحة بكتابة الكلمة `int` في مكان *Variable_Type*، حيث يأخذ متغير من نوع *Integer* مساحة قدرها ٢ بايت و التي تساوي ١٦ بت و تساوي ٦٥٥٣٦ احتمال، أي أن أقصى قيمة يمكن أن يحملها المتغير هي ٦٥٥٣٥، ابتداء من الصفر، أو ابتداء من ٣٢،٧٦٨- إلى ٣٢،٧٦٧ في حالة ضم الأعداد السالبة. و يمكن أن يكون حجمها ٤ بايت (حسب المترجم و نمطه) أي تساوي ٣٢ بت، حيث أقصى قيمة يمكن أن تحملها هي ٤٢٩٤٩٦٧٢٩٦، ابتداء من الصفر(في حالة أن المتغير لا يحتوي إلا على قيم موجبة). مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int Variable_Name = 0;
6  }
```

البرنامج ١,٣,١: طريقة الإعلان عن متغير من نوع عدد صحيح

١,٣,١,٢ متغير الأعداد الحقيقية float:

الأعداد الحقيقية هي الأعداد التي لها فواصل، و يتم الإعلان عنها باستخدام الكلمة `float`، حجمها ٤ بايت، حيث تبدأ من 1.2^{E}-38 إلى 3.4^{E}+38. مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
```

```
5        float Variable_Name = 0.0;
6    }
```

البرنامج ١,٣,٢: طريقة الإعلان عن متغير من نوع عدد حقيقي

١,٣,١,٣ متغير الأعداد الحقيقية double:

double هي ضعف float، و يتم الإعلان عنها باستخدام الكلمة double، حيث حجمها ٨ بايت و تبدأ من 2.3^{E}-308 إلى 1.7^{E}+308. مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        double Variable_Name = 0.0;
6  }
```

البرنامج ١,٣,٣: طريقة الإعلان عن متغير من نوع عدد حقيقي (٢)

١,٣,١,٤ متغير الأعداد الصحيحة short:

هو أيضا من متغيرات الأعداد الصحيحة حيث نقوم بالإعلان عنه بكتابة الكلمة short في مكان *Variable_Type*، حجمه ٢ بايت و التي تساوي ١٦ بت و تساوي ٦٥٥٣٦ احتمال، أي أن أقصى قيمة يمكن أن يحملها المتغير هي ٦٥٥٣٥ ابتداء من الصفر، أو ابتداء من ٣٢،٧٦٨- إلى ٣٢،٧٦٧ في حالة ضم الأعداد السالبة. مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        short Variable_Name = 0;
6  }
```

البرنامج ١,٣,٤: طريقة الإعلان عن متغير من نوع عدد صحيح (٢)

١,٣,١,٥ متغير الأعداد الصحيحة long:

هو أيضا من متغيرات الأعداد الصحيحة حيث نقوم بالإعلان عنه بكتابة الكلمة long في مكان *Variable_Type*، حجمه ٤ بايت أي يساوي ٣٢ بت، حيث أقصى قيمة يمكن أن يحملها هي ٤٢٩٤٩٦٧٢٩٦، ابتداء من الصفر(في حالة أن المتغير لا يحتوي إلا على قيم موجبة). مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        long Variable_Name = 0;
6  }
```

البرنامج ١,٣,٥: طريقة الإعلان عن متغير من نوع عدد صحيح (٣)

١,٣,١,٥ متغير الرموز char:

من أصغر المتغيرات، يتم الإعلان عنه بكتابة الكلمة char في مكان *Variable_Type*، حجمه ١ بايت أي ٨ بت حيث يحمل ٢٥٦ احتمال ابتداء من ٠ إلى ٢٥٥ أو من ١٢٨- إلى ١٢٧، حيث كل رقم يمثل برمز في جدول *ASCII*.

مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5          char Variable_Name = 'A';
6  }
```

البرنامج ١,٣,٦: طريقة الإعلان عن متغير من نوع حرفي

١,٣,٢ اسم المتغير Variable Name:

تحدثنا سابقا عن عناوين، توجد ملاحظة قوية هنا يجب التنبه بها، و هي أننا نرى المتغيرات أسماء، أما الجهاز فيراه عناوين، الإسم مجرد أداة استْعملت لتسهيل عملية الوصول إلى تلك العناوين بدون اللجوء إلى عناوين، إنما أسماء واضحة توضح سبب الإعلان عنها. و لاسم المتغير حدود لا يجب تجاوزها و هي:

- أن لا يتجاوز اسم المتغير أكثر من ٣١ حرف.
- أن لا يبدأ اسم المتغير بأرقام.
- أن لا يكون اسم المتغير يحتوي على مؤثرات مثل الجمع و الطرح و....
- أن لا يكون اسم المتغير يحتوي على رموز مثل % و # و { و...(باستثناء الرمز _).
- أن لا يكون اسم المتغير مستعمل سابقا في دالة أو متغير آخر.
- أن لا يكون اسم المتغير من أسماء الكلمات المحجوزة.

١,٣,٣ قيمة المتغير Variable Value:

يجب مراعاة قيمة المتغير حسب نوعه، فمثلا لا يمكن أن نعطي للمتغير int قيمة عدد حقيقي float. و قيمة المتغير يمكن أن نعطيها له مباشرة بعد الإعلان عنه، أو نقوم بالإعلان عنه و نضع به قيمة فيما بعد(أو نضع به قيمة إستقبلنا من المستخدم مثلا.

١,٣,٤ أمثلة حول المتغيرات:

سأقدم أمثلة مختلفة حول طريقة استعمال المتغيرات، و نبدأ بمتغيرات أعداد صحيحة حيث نقوم بإعلان عن متغير باسم Var و به القيمة ٥، ثم نقوم بطباعة القيمة الموجودة في المتغير Var على الشاشة، المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int Var = 5;
6
7        printf("%d\n", Var);
8  }
```

البرنامج ١,٣,٧: طريقة طباعة محتوى متغير من نوع عدد صحيح

في هذا المثال، في السطر الخامس تم الإعلان عن متغير باسم Var و من نوع int (عدد صحيح) و به القيمة ٥، و في السطر السابع استعملنا الدالة printf لطباعة قيمة المتغير Var، و توجد طرق أخرى لإعطاء للمتغيرات قيم، سأعطي طريقتين، الأولى هي الإعلان عن المتغير في سطر ثم إعطاءه قيمة في سطر آخر مثل:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int Var;
6        Var = 5;
7
8        printf("%d\n", Var);
9  }
```

البرنامج ١,٣,٨: طريقة تحديث قيمة متغير و طبعها

و الطريقة الثانية هي الإعلان عن متغيرين، الأول به القيمة ٥ و الثاني به قيمة المتغير الأول، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int Var_1 = 5;
6        int Var_2 = Var_1;
7
8        printf("%d\n", Var_2);
9  }
```

البرنامج ١,٣,٩: طريقة تحديث قيمة متغير معطاة من متغير آخر

مثال آخر:

```
1  #include<stdio.h>
2
3  main()
```

```
4  {
5        int Num1, Num2, Num3;
6        Num1 = 5;
7        Num2 = 7;
8        Num3 = Num1 + Num2;
9
10       printf("%d + %d = %d\n", Num1, Num2, Num3);
11 }
```

البرنامج ١,٣,١٠: ناتج جمع بين عددين صحيين في متغير

في السطر الخامس تم الإعلان عن ثلاثة متغيرات في نفس السطر حيث نقوم بفصل بين اسم متغير و آخر بفاصلة، و هنا ستكون جميع المتغيرات من نوع أعداد صحيحة (int)، و في السطر السادس و السطر السابع أعطينا للمتغير Num1 القيمة ٥ و المتغير Num2 القيمة ٧، و في السطر الثامن أعطينا للمتغير Num3 نتيجة الجمع بين المتغير Num1 و المتغير Num2، و أخيرا السطر العاشر و الذي يقوم بطباعة نتائج البرنامج. و نفس الطرق السابقة يمكن إستعمالها مع متغيرات من نوع float و short و long، أما المتغير char فطريقة استعماله ليست مختلفة كثير، حيث يمكننا أيضا أن نعطيه عدد بدل الحرف حيث عند طباعته لا يطبع عددا، إنما الحرف الذي يحمل ذلك الرقم في جدول *ASCII*، و لكي تفهم طريقة استعمال متغيرات من نوع char إليك المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        char ch = 'a';
6
7        printf("%c\n", ch);
8  }
```

البرنامج ١,٣,١١: طريقة طباعة حرف موجود في متغير حرفي

في السطر الخامس أعطين للمتغير ch الحرف الذي *a* مع مراعاة أن يكون داخل ' '، أما إذا أردنا أن نعطيه عددا يطبع لنا الحرف *a* فهو الرقم ٩٧ في جدول *ASCII*، و سيصبح المثال السابقة كما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        char ch = 97;
6
7        printf("%c\n", ch);
8  }
```

البرنامج ١,٣,١٢: طريقة طباعة حرف بالإعتماد على رقمه في جدول أسكي

و كما قلنا سابقا أن أقصى قيمة يمكن أن يحملها متغير من نوع char هي ٢٥٥ ابتداء من الصفر، و كل رقم يمثل برمز.

١,٣,٥ الأعداد الموجبة و الأعداد السالبة:

في حالة أنك أردت استعمال أعداد موجبة و أعداد سالبة لمتغير فتوجد طريقتين لذلك الأولى تكون افتراضية عند كتابة نوع و اسم المتغير، أي أنه عندما نقوم بالإعلان عن متغير مثلا int Num يمكنه أن يحمل كلا من الأعداد السالبة و الموجبة، أو يمكن كتابة signed Num و هي مثل int Num من ناحية الحجم و الاستعمال، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5          signed Num = 5;
6
7          printf("%d\n", Num);
8  }
```

البرنامج ١,٣,١٣: متغير ذات إشارة

هذا بالنسبة للمتغيرات التي تحتوي على أعداد موجبة و أعداد سالبة، أما في حالة أردنا أعداد موجبة فقط فسنستعمل الكلمة unsigned قبل اسم المتغير، مثلما هو موضح في المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          unsigned Num = 5;
6
7          printf("%u\n", Num);
8  }
```

البرنامج ١,٣,١٤: متغير بدون إشارة

تم الإعلان عن المتغير الذي لا يحتوي على أعداد سالبة في السطر الخامس و كما أن طريقة إستعمالها كاستعمال متغير طبيعي، و هنا نتعرف على الرمز الجديد %u الذي يخبر المترجم أنا القيمة التي سيتم طبعها من نوع unisgned أي أعداد بدون إشارة.

ملاحظة:

عند الإعلان عن متغير طبيعي مثلا char فإنه سيحتوي ٢٥٦ احتمال كما قلنا سابقا،حيث يمكن أن يبدأ من ٠ إلى ٢٥٥ في حالة عدم الحاجة إلى أعداد سالبة، و يمكن أيضا أن يبدأ من ١٢٨- إلى ١٢٧ في حالة استعمال أعداد سالبة، و لكي تعرف السبب في ذلك إليك الشرح:

لكل متغير حجمه، مثلا متغير من نوع char حجمه ١ بايت أما int فحجمه هو ٢بايت، و كما نعرف أن ١ بايت يساوي ٨ بت، أي أن متغير من نوع char حجمه ٨ بت، و ١ بت يساوي إما ١ أو ٠ حيث هذا يعني أن ١ بت

لديه احتمالين (٠ أو ١)، أما ٨ بت فلديها ٢٥٦ احتمال تبدأ من ٠ و تنتهي عند ٢٥٥ إن لم تكن تحتوي على أعداد سالبة، أما في حالة أردنا أعداد سالب فسيتم سحب ١ بت من ٨، أي سيصبح حجم متغير من نوع char ٧ بت أمـــا البت الثامن سنتركه للإشارة، و سيحمل إما الإشارة + أو الإشارة -، و ٧ بت تساوي ١٢٨ احتمال. و هذا جدول به القيم المحتملة لكل نوع من المتغيرات:

النوع	الحجم	القيم
char و signed char	١ بايت	من -١٢٧ إلى ١٢٧
char و unsigned char	١ بايت	من ٠ إلى ٢٥٥
int و signed int	٢ بايت	من -٣٢،٧٦٨ إلى ٣٢،٧٦٧
int و unsigned int	٢ بايت	من ٠ إلى ٦٥٥٣٥
short و signed short	٢ بايت	من -٣٢،٧٦٨ إلى ٣٢،٧٦٧
short و unsigned short	٢ بايت	من ٠ إلى ٦٥٥٣٥
long و signed long	٤ بايت	من -٢،١٤٧،٤٨٣،٦٤٨ إلى ٢،١٤٧،٤٨٣،٦٤٧
long و unsigned long	٤ بايت	من ٠ إلى ٤،٢٩٤،٩٦٧،٢٩٥

الجدول ١,٣,١: أنواع المتغيرات و أحجامها

الثوابت، هي عكس المتغيرات، يمكن أن تكون عدد، حرف، أو نص، حيث لا يمكن التغير قيمتها أي تـــصبح قابلـــة للقراءة فقط، سأعطي مثال، حيث هذا مثال به متغير من نوع أعداد صحيحة، و نعطيه القيمة ٥ ثم نقوم بطبع المـــتغير على الشاشة ثم نقوم بتحديث المتغير إلى القيمة ٨ ثم نعيد طباعة قيمة المتغير و ها هو المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int num;
6
7        num = 5;
8
9        printf("%d\n", num);
10
11       num = 8;
12
13       printf("%d\n", num);
14 }
```

البرنامج ١,٣,١٥: طريقة تحديث قيمة متغير

هنا سيتم تغير قيمة المتغير num من ٥ إلى ٨، و هذه الطريقة صحيح. و الآن سنكتب نفس البرنامج السابق مع إضـــافة بسيطة، المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        const int num;
6
7        num = 5;
8
9        printf("%d\n", num);
10
11       num = 8;
12
13       printf("%d\n", num);
14 }
```

البرنامج ١,٣,١٦: طريقة الإعلان عن ثابت و التحديث في قيمته

الإضافة موجودة في السطر الخامس، و هي إضافة الكلمة const إلى المتغير num int و التي تعني أن المتغير num ثابت، و هنا البرنامج لن يعمل و السبب هو أنه لا يمكن تحديث القيمة الأولى لثوابت، و في مثالنا السابقة لا توجد قيمة للمتغير num بعدما أن تم الإعلان عنه، و يجب دائما إعطاء قيم لثوابت مباشرة بعد الإعلان عنها و إلا ستكون عبارة عن ثوابت ذات أعداد عشوائية ثابتة لا يمكن التحديث فيها، و هذا المثال السابق بعد التصحيح:

```
1  #include<stdio.h>
2
3  main()
4  {
5        const int num = 5;
6
7        printf("%d\n", num);
8  }
```

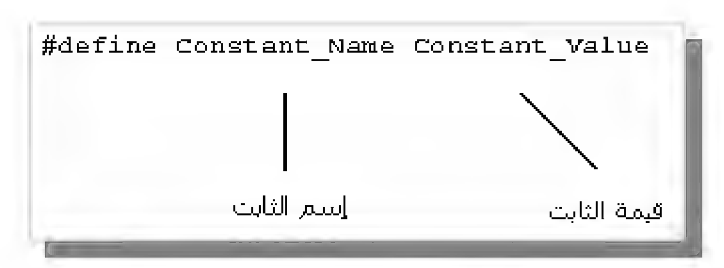
البرنامج ١,٣,١٧: طريقة الإعلان عن ثابت

و يمكن أيضا كتابة الكلمة const مباشرة بعد نوع المتغير مثل:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int const num = 5;
6
7        printf("%d\n", num);
8  }
```

البرنامج ١,٣,١٨: طريقة الإعلان عن ثابت (٢)

و يمكن استعمال نفس أنواع المتغيرات على الثوابت. و يوجد نوع آخر من الثوابت، و هي باستعمال الكلمة `#define` و طريقة إستعمالها موضحة كما في الصورة التالية:



الشكل ١,٣,٣: طريقة الإعلان عن ثابت

حيث سيصبح المثال السابقة كما يلي:

```
1  #include<stdio.h>
2
3  #define num 5
4
5  main()
6  {
7        printf("%d\n", num);
8  }
```

البرنامج ١,٣,١٩: طريقة الإعلان عن ثابت (٣)

تم الإعلان عن الثابت `num` في السطر الثالث، ثم طباعة قيمته في السطر السابع. و كما قلت سابقا، أن *preprocessor* تبدأ بالرمز #، و هذا يعني أن `#define` من الـ *preprocessors*، و جميع *preprocessors* لا تنتهي بفواصل منقوطة.

١,٣,٦ الأخطاء المحتملة:

١. لا يمكن وضع قيمة أكثر من قيمة المتغير القصوى.
٢. لا يمكن الإعلان عن المتغيرات إلا في بداية كل *block*.
٣. في حالة لم يتم تعين قيمة لمتغير، و أردت طبع قيمة ذلك المتغير على الشاشة فستأتي أعداد عشوائية تختلف من جهاز لآخر.
٤. يجب الحرص على أن بين نوع المتغير و اسم المتغير مسافة واحدة على الأقل.
٥. لا يمكن كتابة الكلمة `const` بعد اسم المتغير أو بعد الإعلان عنه.
٦. لا يمكن تغير اسم متغير أو ثابت.
٧. لا يمكن الإعلان عن متغيرين بنفس الاسم.

١,٣,٧ تمارين:

١. أكتب برنامج يقوم بطباعة العديدين ٣,١٤ و ١٥، باستخدام الرموز الخاصة بطباعتها.

٢. ماذا يحدث إن أعطين لمتغير من نوع int قيمة أكثر من ٦٥٥٣٥ ؟

٣. أكتب برنامج يقوم بطباعة الحرف *A* بدل الرقم ٦٥، بدون استخدام متغيرات أو ثوابت.

٤. أكتب برنامج يقوم بطباعة الحرف *A* بدل الرقم ٦٥، باستخدام char.

٥. أكتب برنامج به ثلاثة متغيرات، المتغير الأول به القيمة ١٨ و الثاني ٨٩، أما الثالث يكون الناتج الحاصل بين المتغير الأول و الثاني في كل من الجمع، الطرح، القسمة و الضرب.

٦. أكتب برنامج به ثلاثة من #define *preprocessor* حيث الثالثة هي نتيجة الجمع بين الأولى و الثاني، الأولى بها القيمة ٥ و الثاني بها القيمة ١٥.

١,٤ التعليقات Comments

التعليقات هي مجموعة من سلاسل نصية نستعملها لتوضيح أوامر في مصادر برامجنا، و يمكن أن تحتـــوي تلــك النصوص على أرقام، أحرف، أو رموز يقوم المترجم بتجاهلها.

١,٤,١ فائدة التعليقات:

فائدة التعليقات يمكنك أن تلاحظها في الكثير من الأمثل المفتوحة المصدر الموجودة على الإنترنت، مثلا تجــد مثــال لبرنامج ما كبير و غير واضح، و هنا يلجئ المبرمج إلى استعمال التعليقات لجعلها أكثر وضوح.

١,٤,٢ أنواع التعليقات:

يوجد نوعين من التعليقات هما:

١,٤,٢,١ التعليقات بالنصوص الطويلة:

التعليقات بالنصوص الطويلة هي نصوص بها أكثر من سطر، و طريقة استخدامها هي تحديد بداية التعليق و التي تبدأ بـــ /* و نضع */ في نهاية التعليق، مثال:

```
1  /*
2  My First Program:
3  Hello, World!
4  */
5
6  #include<stdio.h> /* Standart Input Output Header File*/
7
8  main()                     /*main function*/
9  {/*Start of main function*/
10       printf("Hello, World!");/*to print Hello, World!*/
11 }/*End of main function*/
```

البرنامج ١,٤,١: التعليقات بالنصوص الطويلة

١,٤,٢,٢ التعليقات بالأسطر:

التعليقات بالأسطر هي تجاهل السطر التعليقي حيث تبدأ بـــ //، تمت إضافتها في لغة *C++* القياسية، هـــذا مثــال يوضح طريقة إستعمالها:

```
1  //My First Program:
2  //Hello, World!
3
```

```
4  #include<stdio.h> //Standart Input Output Header File
5
6  main()                    //main function
7  {//Start of main function
8        printf("Hello, World!");//to print Hello, World!
9  }//End of main function
```

البرنامج ١,٤,٢: التعليقات السطرية

و هذه الطريقة ليست من طرق لغة C القياسية في التعليقات، و لكن الكثير من المترجمات تدعمها.

١,٤,٣ كيف يستعمل المبرمجون التعليقات:

لا تستعمل التعليقات في تبين أو توضيح تعليمات في البرامج، فمثلا يستعمل مبرمجون آخرون التعليقات لتبين اسم الملف، كاتبه، شرح مختصر ثم تاريخ إنشاء الملف، مثال:

```
1  /*******************************************************************************/
2  /*author          : Khalil Ounis                                                */
3  /*Date            : 2006/01/01                                                  */
4  /*Information     : This is small program show                                  */
5  /*                  how programmers use comments                                */
6  /*            All rights reserved (c) 2006/2007                                 */
7  /*******************************************************************************/
8
9  /*******************************************************************************/
10 /*Function   : Principal function                                               */
11 /*Input      : None                                                             */
12 /*Output     : None                                                             */
13 /*******************************************************************************/
14 main()
15 {
16 /*Empty project*/
17 }/*Main function end*/
```

البرنامج ١,٤,٣: كيفية استعمال التعليقات

١,٤,٤ الأخطاء المحتملة:

١. في التعليقات بالنصوص الطويلة إن لم يتم تحديد نهاية التعليق فإن كل ما هو بعد بداية التعليق يعتبر تعليق، و هذا مثال توضيحي:

```
1  /*comment
2  #include<stdio.h>
3
4  main()
5  {
6        printf("Hello, World!");
7  }
```

البرنامج ١,٤,٤: الخطأ ١

هنا البرنامج كله عبارة عن تعليق، أي أن المشروع فارغ.

٢. في التعليقات السطرية يجب الانتباه إلى ما نضعه تعليق فمثلا:

```
1 //#include<stdio.h>
2 
3 main()
4 {
5 	printf("Hello, World!");
6 }
```

البرنامج ١,٤,٥: الخطأ ٢

هنا سيخبرك المترجم على أن الدالة printf غير معرفة سابقا، و هذا الخطأ سببه هو جعل الكلمة المحجوزة #include الخاصة بضم الملف الرأسي stdio.h عبارة عن تعليق، و هنا سيتم تجاهل ضم الملف الرأسي stdio.h.

١,٤,٥ تمارين:

لا توجد تمارين في هذا الدرس.

١,٥ الإدخال Input

في الدروس السابقة لم ندرس إلا الإخراج باستخدام الدالة printf، الآن سنعرف كيفية الإدخال بواسطة الدالة دوال خاصة بالإخراج، حاليا سنرى الدالة scanf، التشابه كبير جدا بين الداليتين printf و scanf، فقط الأولى خاصة بالإخراج و الثانية خاصة بالإدخال. تستعمل الدالة scanf لقراءة أو استقبال المعلومات من أداءه الإدخال لوحة المفاتيح *keyboard*. الآن سنقوم بكتابة برنامج يطلب من المستخدم إدخال قيمة ثم نعطيه القيمة التي قام بإدخالها:

```
1  /*الإدخال*/
2  #include<stdio.h>
3
4  main()
5  {
6        int usr_val;          /*هنا سنضع القيمة التي سيدخلها المستخدم*/
7
8        printf("Enter a value: ");/*هنا نطلب من المستخدم إدخال قيمة*/
9        scanf("%d", &usr_val); /*يقموم الجهاز بإنتظار دخول قيمة من المستخدم*/
10       printf("Your value is: %d\n", usr_val);/*و هنا نطبع القيمة التي أدخلت*/
11 }
```

البرنامج ١,٥,١: طريقة إستعمال الدالة scanf لإدخال قيمة صحيحة

البرنامج موضح بالتعليقات، في السطر التاسع قمنا باستخدام الدالة scanf و داخلها ستجد وسيطين، الأول نقوم فيه بتحديد نوع القيمة التي سيدخلها المستخدم حيث هنا وضعنا الرمز %d و الذي درسنه سابقا في الدالة printf حيث قلنا أنها خاص بالأعداد الصحيحة، و في الوسيط الثاني يوجد &usr_val، و الرمز & يعني وضع القيمة التي أدخلها المستخدم في عنوان المتغير usr_val، و ستفهم السبب إضافة الرمز & في الدروس القادمة. ، إلا هنا تكون قيمة usr_val قد أصبحت القيمة التي أدخلها المستخدم ثم نقوم بطباعتها على الشاشة.

المثال السابق خاص بإدخال الأعداد الصحيحة، أما بالنسبة لباقي أنواع المتغيرات فسنستعمل نفس الطريقة فقط نقوم بتغير الرمز %d إلى نوع المتغير الذي نريد إستقباله، فمثلا إذا أردنا من المستخدم أن يقوم بإدخال رمز بدل رقم نضع الرمز %c في الدالة scanf، و هذا مثال يوضح ذلك:

```
1  #include<stdio.h>
2
3  main()
4  {
5        char usr_char;
6
7        printf("Enter a character: ");
8        scanf("%c", &usr_char);
9        printf("Your character is: %c\n", usr_char);
10 }
```

البرنامج ١,٥,٢: طريقة إستعمال الدالة scanf لإدخال حرف

الآن سنقوم بكتابة برنامج يطلب من المستخدم إدخال قيمة، ثم يطلب منه إدخال قيمة ثانية، و نعطيه النتائج بجميع المؤثرات الأساسية:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int val1, val2;
6
7        printf("1)Enter a value: ");
8        scanf("%d", &val1);
9
10       printf("2)Enter a value: ");
11       scanf("%d", &val2);
12
13       printf("%d + %d = %d\n", val1, val2, val1+val2);
14       printf("%d - %d = %d\n", val1, val2, val1-val2);
15       printf("%d * %d = %d\n", val1, val2, val1*val2);
16       printf("%d / %d = %d\n", val1, val2, val1/val2);
17 }
```

البرنامج ١,٥,٣: طريقة إستعمال الدالة scanf لإدخال قيمة صحيحة (٢)

المثال واضح، قمنا بالإعلان عن متغيرين في السطر الخامس، ثم طلبنا من المستخدم إدخال قيمة في السطر السابع و قمنا بأخذ القيمة في السطر الثامن، و بعدها طلبنا من المستخدم إدخال القيمة الثانية ثم أخذنا القيمة الثانية في السطر الحادي عشر، ثم طبعنا النتائج في كل من السطر ١٣، ١٤، ١٥ و ١٦. و يمكن استعمال إدخال متعدد في الدالة scanf، و هذا المثال السابقة باستخدام الإدخال المتعدد:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int val1, val2;
6
7        printf("Enter two value: ");
8        scanf("%d%d", &val1, &val2);
9
10       printf("%d + %d = %d\n", val1, val2, val1+val2);
11       printf("%d - %d = %d\n", val1, val2, val1-val2);
12       printf("%d * %d = %d\n", val1, val2, val1*val2);
13       printf("%d / %d = %d\n", val1, val2, val1/val2);
14 }
```

البرنامج ١,٥,٤: طريقة إستعمال الدالة scanf لإدخال قيمة صحيحة (٣)

التعدد هنا موجود في السطر الثامن، في الدالة scanf، و يمكن أن نزيد أكثر من ذلك، فقط نضيف الرمز الخاص بنوع المتغير ثم إضافة اسم المتغير في آخر الدالة.

١,٥,١ الأخطاء المحتملة:

١. لا يمكن استخدام الوسيط الأول من الدالة scanf لطباعة(الإخراج)، الوسيط الأول من هذه الدالة خاص بنوع الرموز التي سيستقبلها البرنامج من المستخدم.

٢. في حالة الاستغناء عن الرمز & فستكون النتائج غير صحيحة.

٣. يجب الانتباه إلى عدد المتغيرات المراد فحصها، يجب أن يكون عدد رموز أنواع المتغيرات نفسه عدد المـــتغيرات المراد فحصها.

١,٥,٢ تمارين:

١. أكتب برنامج يطلب من المستخدم إدخال الحرف الأول و الأخير من اسمه ثم يقوم البرنامج بإخبار المستخدم أن إسم يبدأ بـــ "الحرف الأول الذي أدخله المستخدم" و ينتهي بالحرف "الحرف الأخير الذي أدخله المستخدم".

١,٦ المؤثرات Operators

للمؤثرات أنواع أهمهما ثلاثة و هي:

١,٦,١ المؤثرات الحسابية (arthimetic operators):

هي المؤثرات الحسابية الأساسية و التي تتمثل في كل من الجمع(+)، الطرح(-)، القسمة(/) و الضرب(*)،و توجد مؤثرات أخرى خاص بلغة C و هي: الزيادة(++)، النقصان(--) و باقي القسمة(%).

بالنسبة للجمع، الطرح، القسمة و الضرب فقد أخذنا أمثلة عنها سابقا. سنأخذ أمثلة عن كل من مؤثرات الزيادة و النقصان و باقي القسمة:

١,٦,١,١ مؤثر الزيادة increment (++):

تعتبر مؤثرات الزيادة و النقصان من أهم المؤثرات، تستعمل في الكثير من البرامج و خاصة في حلقات التكرار(سندرسها في الدروس القادمة). مؤثر الزيادة يعني زيادة رقم واحد إلى المتغير الذي نريد الزيادة إليه، و هذا مثال يوضح ذلك:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int Inc;
6        Inc = 0;
7
8        printf("Inc = %d\n", Inc);
9
10       Inc++;
11
12       printf("Inc = %d\n", Inc);
13
14       ++Inc;
15
16       printf("Inc = %d\n", Inc);
17 }
```

البرنامج ١,٦,١: طريقة إستعمال مؤثر الزيادة

قمنا باستعمال مؤثر الزيادة في كلا من السطر العاشر و السطر الرابع عشر، و نتائج البرنامج تكون ٠ ثم ١ ثم ٢. و يمكن استعمال طرق أخرى مثل:

```
1  printf("Inc = %d\n", Inc);
2
```

```
3 Inc = Inc+1;
4
5 printf("Inc = %d\n", Inc);
6
7 Inc += 1;
8
9 printf("Inc = %d\n", Inc);
```

البرنامج ١,٦,٢: طريقة إستعمال مؤثر الزيادة (٢)

الطريقة الأولى هي `Inc+1 = Inc` و التي موجود في السطر الثالث، هي مطابقة تماما لـ `Inc++` و لكن في السابقة يمكن أن نقوم بزيادة أكثر من ١ يعني إن كتبنا `Inc+3 = Inc` فسيتم الإضافة إلى المتغير `Inc` ثلاثة أرقام، كذلك الطريقة الثانية `Inc += 1`، هي مثل `Inc+1 = Inc` تمام. و لكن يستحسن دائما استعمال ++ عند الزيادة بالواحد، و في حالة أن زيادة ستكون أكثر من واحد فسنستعمل الطرق الأخرى. و الفرق بين أن يكون المؤثرين ++ في بداية اسم المتغير أو نهايته هو موضح في المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int Inc;
6        Inc = 0;
7
8        printf("Inc = %d\n", Inc);
9        printf("Inc = %d\n", Inc++);
10       printf("Inc = %d\n", Inc);
11       printf("Inc = %d\n", ++Inc);
12       printf("Inc = %d\n", Inc);
13 }
```

البرنامج ١,٦,٣: طريقة إستعمال مؤثر الزيادة (٣)

في السطر الثامن سيتم طباعة العدد ٠، و في السطر التاسع أيضا و معنا ذلك عند كتابة متغير قم المؤثر ++ يعني طباعته ثم تنفيذ مؤثر التزايد. و في السطر العاشر سيتم طبع العدد ١ لأننا قمنا بالزيادة في السطر التاسع. و في السطر الحادي عشر سيتم تنفذ مؤثر التزايد أولا يم طباعة النتيجة التي هي ٢، و كذلك في السطر الثاني عشر سيتم طباعة العدد ٢.

١,٦,١,٢ مؤثر النقصان **decrement** (--):

سنتعامل مع المؤثر النقصان مثلما تعاملنا مع مؤثر الزيادة فقط بدل ++ نضع -- و هذا المثال السابق باستخدام مؤثر النقصان:

```
1  #include<stdio.h>
2
3  main()
```

```
4  {
5         int Dec;
6         Dec = 2;
7  
8         printf("Dec = %d\n", Dec);
9  
10        Dec--;
11 
12        printf("Dec = %d\n", Dec);
13 
14        --Dec;
15 
16        printf("Dec = %d\n", Dec);
17 }
```

البرنامج ١,٦,٤: طريقة إستعمال مؤثر النقصان

و يمكن أيضا استعمال نفس الطرق السابقة في النقصان:

```
1 printf("Dec = %d\n", Dec);
2 
3 Dec = Dec-1;
4 
5 printf("Dec = %d\n", Dec);
6 
7 Dec -= 1;
8 
9 printf("Dec = %d\n", Dec);
```

البرنامج ١,٦,٥: طريقة إستعمال مؤثر النقصان (٢)

١,٦,١,٣ مؤثر باقي القسمة (%):

أيضا يمكن اعتباره من المؤثرات المهمة، و طريقة استعماله سهل فمثلا لو أردنا أن نجد باقي القسمة بين العـــدد ٥ و العدد ٣ نكتب *5%3*، و هذا مثال توضيحي:

```
1 #include<stdio.h>
2 
3 main()
4 {
5        printf("%d\n", 5%3);
6 }
```

البرنامج ١,٦,٦: طريقة إستعمال مؤثر باقي القسمة

١,٦,٢ المؤثرات العلاقية (**relational operators**):

هي مؤثرات تعتمد على المقارنة بين قيمة و قيمة أخرى، حيث تكون النتيجة إما صحيحة (*true*) أو خاطئة (*false*)، و هذا مثال يوضح ذلك:

```
1 #include<stdio.h>
```

```
2
3 main()
4 {
5       printf("%d\n", 5<3);
6       printf("%d\n", 5==3);
7       printf("%d\n", 5>3);
8 }
```

البرنامج ١,٦,٧: طريقة إستعمال المؤثرات العلاقية

هنا ستجد نتائج البرنامج:

٠: لأن العدد ٥ ليس أقل من ٣.

٠: لأن العدد ٥ لا يساوي العدد ٣.

١: لأن العدد ٥ أكبير من ٣.

حيث ٠ تعني خطأ *(true)* و ١ تعني صحيح *(false)*، و أيضا ستلاحظ أنه كتبنا في السطر السادس ٥==٣ و ليس ٥=٣ و ذلك لأننا نقارن و عند المقارنة نكتب ==، و إذا وضعت = مكان == فسيخبرك المترجم عن وجود خطأ. و يوجد كذلك المؤثر أكبر من أو يساوي و يمثل في لغة *C* بـ =>، و مؤثر أصغر من أو يساوي بـ =<، و المؤثر لا يساوي بـ *=!*.

١,٦,٣ المؤثرات المنطقية (**logical operators**):

و هي مؤثرات تعتمد على المؤثرات العلاقية في نتيجتها و لكن لها رموزها الخاصة و هي:

&& و التي تعني "و"

|| و التي تعني "أو"

! و التي يعني "لا"

و هذا مثال يوضح ذلك:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("%d\n", 5<3 && 5>3);
6        printf("%d\n", 5==3 && 3==5);
7        printf("%d\n", 5>3 && 5<3);
8
9        printf("%d\n", 5<3 || 5>3);
10       printf("%d\n", 5==3 || 3==5);
11       printf("%d\n", 5>3 || 5<3);
12
```

```
13        printf("%d\n", !(5<3));
14        printf("%d\n", !(5==3));
15        printf("%d\n", !(5>3));
16  }
```

البرنامج ١,٦,٨: طريقة إستعمال المؤثرات المنطقية

نتائج البرنامج هي:

٠: لأنه توجد علاقة خاطئة و هي 5<3.

٠: لأن كلا العلاقتين خطاءتين.

٠: لأنه توجد علاقة خاطئة و هي 5<3.

١: لأنه توجد علاقة صحيحة و هي 5>3.

٠: لأن كلا العلاقتين خطاءتين.

١: لأنه توجد العلاقة صحيحة و هي 5>3.

١: لأن ٥ ليست أصغير من ٣.

١: لأن ٥ لا تساوي ٣.

٠: لأن ٥ أكبر من ٣.

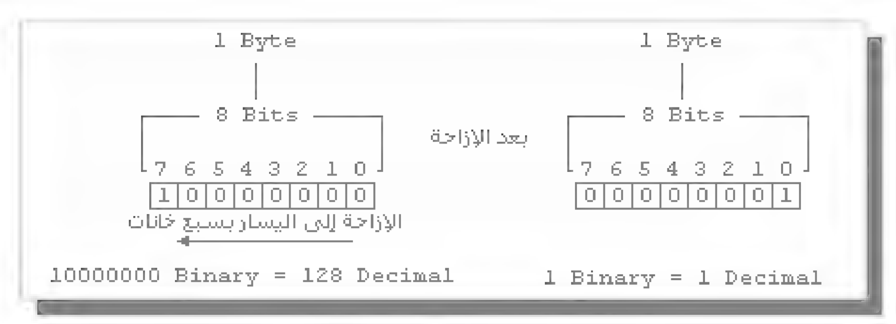
١,٦,٤ مؤثرات أخرى:

توجد مؤثرات أخرى خاص بلغة *C* منها: المؤثر << و يسمى بمؤثر الإزاحة اليسار *Left Shift*، و هي تقوم بإزاحة بت واحدة من بيات (أو بيتات) إلى اليسار، مثال توضيحي لطريقة استعماله:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("%d\n", 1<<7);
6  }
```

البرنامج ١,٦,٩: مؤثر الإزاحة إلى اليسار

هنا الناتج سيكون ١٢٨، و الصورة التالية توضح ذلك:



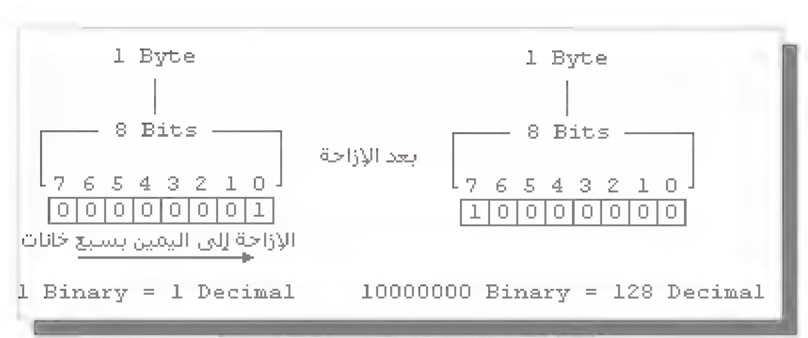
الشكل ١,٦,١: الإزاحة إلى اليسار

المؤثر المعاكس لسابق هو >>، و يسمى بمؤثر الإزاحة إلى اليمين *Right Shift*، و هذا مثال توضيحي لطريقة استعماله:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("%d\n", 128>>7);
6  }
```

البرنامج ١,٦,١٠: مؤثر الإزاحة إلى اليمين

صورة توضيحية:



الشكل ١,٦,٢: الإزاحة إلى يمين

المؤثر #، و هو يستعمل مع التوجيه #define، مثال:

```
1  #include<stdio.h>
2
3  #define printer(str) printf(#str "\n")
4
5
6  main()
7  {
8        printer(Hello);
9  }
```

البرنامج ١,٦,١١: طريقة إستعمال المؤثر #

المؤثرين ## معا، أيضا يتم التعامل معهما مع التوجيه define#، و يمكن تسميتهما بالــ *Merge* أي الدمج، مثال:

```
1  #include<stdio.h>
2
3  #define MergingName(str, strplus) str##strplus
4
5  main()
6  {
7      char MergingName(ch, 1) = 'H';
8      char MergingName(ch, 2) = 'e';
9      char MergingName(ch, 3) = 'l';
10     char MergingName(ch, 4) = 'l';
11     char MergingName(ch, 5) = 'o';
12
13     printf("%c%c%c%c%c\n", ch1, ch2, ch3, ch4, ch5);
14
15 }
```

البرنامج ١,٦,١٢: طريقة إستعمال المؤثرين ##

١,٦,٥ مؤثرات خاصة بالبتات (*bitwize*):

هي مجموعة من المؤثرات خاصة بالبتات *bits*، تقوم بالمقارنة بين بتات بدل البيتات، و هي مشابه للمؤثرات السابقة.

نبدأ بالمؤثر "أو" *OR* و الذي هو الرمز | وحده، و قلنا سابقا أن المؤثر "أو" يحتاج إلى شرط واحد صــحيح، صــورة توضيحية:



الشكل ١,٦,٣: إستعمال المؤثر أو | OR

العملية هي إذا كان بت به القيمة ٠ يقارن مع بت به قيمة ٠ فسيكون الناتج صفر لأن كلا البتات يحتوي على القيمـــة ٠ أي خطأ، و إذا كان بت به القيمة ٠ يقارن مع بت به القيمة ١ فسيكون الناتج واحد لأنه توجــد القيمـــة ١ أي صحيح حيث قلنا أن المؤثر "أو" يحتاج إلى قيمة صحيحة واحدة حتى يكون الناتج صحيح. مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5      int x = 1; /* 1 decimal = 00000001 Binary*/
6      x = x|4;   /* 4 decimal = 00000100 Binary*/
7                 /*           = 00000101 Binary, x = 5 decimal*/
8      printf("x = %d\n", x);
9  }
```

البرنامج ١,٦,١٣: طريقة إستعمال المؤثر أو | OR

بعد هذا المؤثر يأتي المؤثر "و" *AND* و يكون بإستخدام الرمز &، صورة توضيحية:



الشكل ١,٦,٤: إستعمال المؤثر و & AND

يحتاج المؤثر "و" إلى أن تكون كلا القيمتين صحيحتين، حيث إذا كانت قيمة بت ٠ و قيمة البت الثـاني ٠ فـسيكون الناتج ٠ لأن كلا بتات خاطئين، و إذا كانت قيمة بت ١ و الآخر ٠ فأيضا ستكون النتيجة ٠ أي خاطئة لأن المـؤثر "و" يحتاج إلى قيمتين صحيحتين، أما إذا كان بت ١ و الآخر ١ فستكون النتيجة هنا ١ أي صحيح. مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5       int x = 1; /* 1 decimal = 00000001 Binary*/
6       x = x&4;   /* 4 decimal = 00000100 Binary*/
7                  /*           = 00000000 Binary, x = 0*/
8       printf("x = %d\n", x);
9  }
```

البرنامج ١,٦,١٤: طريقة إستعمال المؤثر و & AND

و بعد هذا المؤثر سنرى المؤثر *XOR*، و هو ليس كسابقه، و لكنه مشابه للمؤثر *OR* إلا أنه سيكون الناتج ٠ إذا كـان كلا البتات به القيمة ١، و ذلك عبر الرمز ^، صورة توضيحية:



الشكل ١,٦,٥: إستعمال المؤثر ^ XOR

هنا إذا كان بت به القيمة ٠ و الثاني به القيمة ٠ فسيكون الناتج ٠، و إذا كان بت به قيمة ١ و الآخر بـــه القيمـــة ٠ فسيكون الناتج ١، أما إذا كان كلا البتات به القيمة ١ فسيكون الناتج ٠ أي خاطئ و هذا هو الفرق بين هذا المؤثر و مؤثر *OR*. مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int x = 4; /* 4 decimal = 00000100 Binary*/
6          x = x^4;   /* 4 decimal = 00000100 Binary*/
7                     /*           = 00000000 Binary, x = 0*/
8          printf("x = %d\n", x);
9  }
```

البرنامج ١,٦,١٥: طريقة إستعمال المؤثر ^ XOR

و أخيرا المؤثر "لا" *NOT* عبر الرمز ~، و لا يستعمل هذا المؤثر لوحده، و يستعمل بكثر مع المؤثر "و" *AND* لتثبيت بت ما على ٠ بدون التأثير على باقي البتات. و مثالا على ذلك إذا كان لدين القيمة ١ في بت ثالث نريد تصفيره فسيكون ذلك كتالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int x = 5; /* 5 decimal = 101 Binary*/
6          x = x& ~4; /* 4 decimal = 100 Binary*/
7                     /*           = 001 Binary, x = 1*/
8          printf("x = %d\n", x);
9  }
```

البرنامج ١,٦,١٦: طريقة إستعمال المؤثر لا ~ NOT

١,٦,٦ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

١,٦,٧ تمارين:

١. هل يمكن أن نقوم بالتزايد و التناقص داخل الدالة printf ؟
٢. ماذا تعني القيمة ٠ و القيمة ١ في المؤثرات العلاقية ؟
٣. ماذا تعني كل من && و || و ! ؟

١,٧ القرارات if, else, else...if

تستعمل القرارات (و يمكن تسميتها بالجمل الشرطية) للمقارنة بين علاقة حيث تكون إما صحيحة أو خاطئة، في كل من حالة لها أوامر خاصة نقوم بتحديدها، مثلا إذا كانت المقارنة صحيح فسيتم تنفيذ الأوامر التي حددناها في حالة أن المقارنة صحيحة، أما إذا كانت المقارنة خاطئة فسيتم تنفيذ الأوامر المعاكسة.

١,٧,١ استعمال `if`:

لكي تفهم طريقة استعمال `if` ألقي نظرة على الصورة التالية:

```
if ( المقارنة، أو الشرط )          أو          if ( المقارنة، أو الشرط )
{                                                  الأمر;
    الأمر;
}
```

الشكل ١,٧,١: طريقة إعلان شرط ذات أمر واحد

الصورة السابقة يوجد بها مثال بدون الرمز { و } لأنه لا يوجد سوى أمر واحد لتنفيذه، و هذه صورة في حالة وجـــود أكثر من أمر:

```
if ( المقارنة، أو الشرط )
{
        الأمر 1;
        الأمر 2;
        ........;
        الأمر س;
}
```

الشكل ١,٧,٢: طريقة إعلان شرط ذات عدة أوامر

مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int usr_val;
6
7        printf("Enter a value: ");
8        scanf("%d", &usr_val);
```

```
9
10        if(usr_val <100)
11        {
12              printf("%d are less than 100\n", usr_val);
13        }
14
15        if(usr_val >100)
16        {
17              printf("%d are great than 100\n", usr_val);
18        }
19  }
```

البرنامج ١,٧,١: طريقة إستعمال if

في هذا البرنامج طلبنا من المستخدم إدخال قيمة، ثم استعملنا الشرط أو المقارنة في السطر العاشر و السطر الخامس عشر و هو المقارنة بين القيمة التي أدخلها المستخدم و ١٠٠، إذا كانت المقارنة صحيحة فسيقوم البرنامج بتنفيذ ما هو داخل الـ *block* الخاص بـ if، و في حالة أن المقارنة خاطئة فسيتم تجاهل ما هو داخل الـ *block* الذي يلي if. و يمكن كتابة البرنامج بهذه الطريقة:

```
1   #include<stdio.h>
2
3   main()
4   {
5         int usr_val;
6
7         printf("Enter a value: ");
8         scanf("%d", &usr_val);
9
10        if(usr_val <100)
11              printf("%d are less than 100\n", usr_val);
12
13        if(usr_val >100)
14              printf("%d are great than 100\n", usr_val);
15  }
```

البرنامج ١,٧,٢: طريقة إستعمال if (٢)

لأنه في كلا الشرطين لا يوجد بهما إلا أمر واحد.

١,٧,٢ استعمال else:

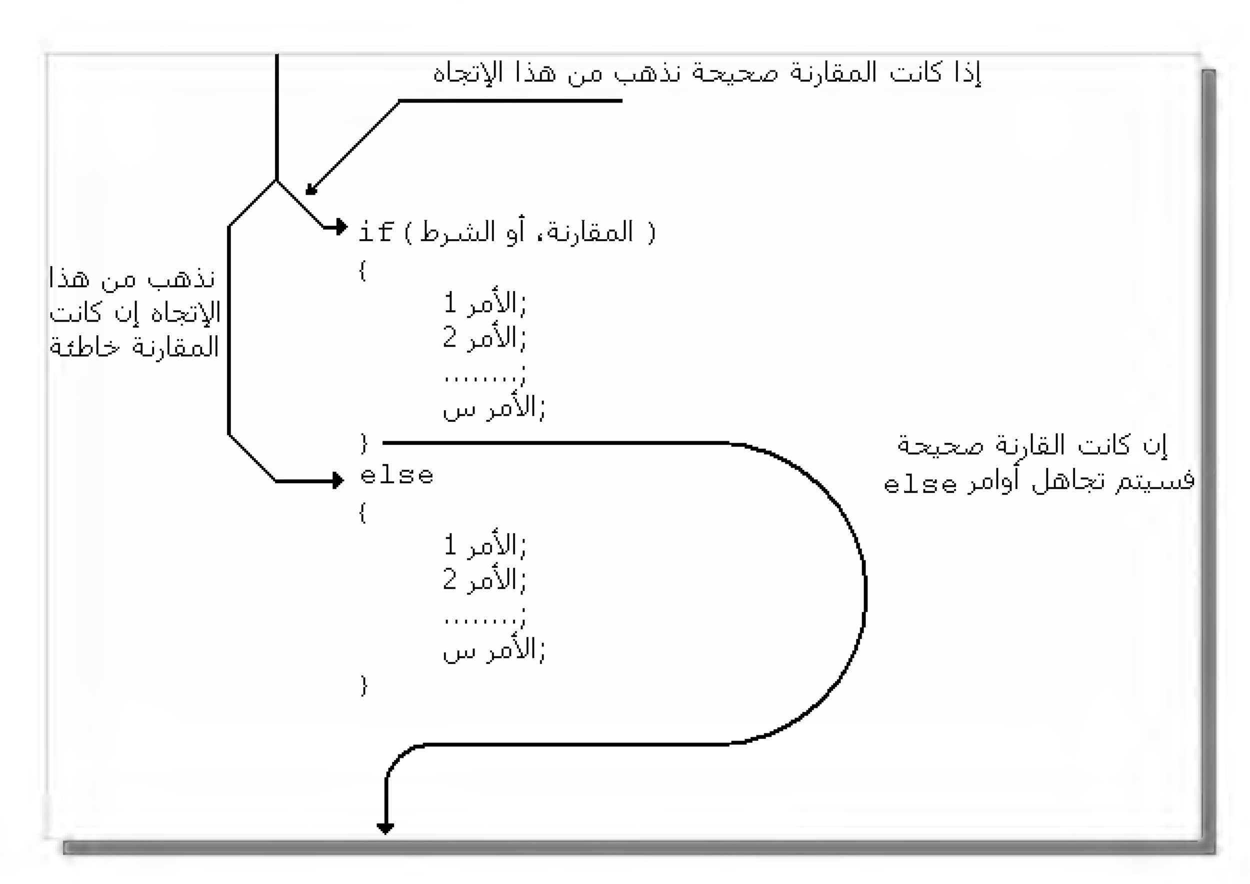
تستعمل الكلمة else مع الكلمة if دائما حيث لا يمكن استعمالها لوحدها، سنكتب الآن البرنامج السابق باستخدام else:

```
1   #include<stdio.h>
2
3   main()
4   {
```

```
5        int usr_val;
6
7        printf("Enter a value: ");
8        scanf("%d", &usr_val);
9
10       if(usr_val<100)
11           printf("%d are less than 100\n", usr_val);
12       else
13           printf("%d are great than 100\n", usr_val);
14   }
```

البرنامج ١,٧,٣: طريقة إستعمال else

استعملنا else في السطر الثاني عشر و سترى أن المثال سيعمل مثل المثال السابق و هنا تجد فائدة else. إن لم تكـن المقارنة صحيح في if فسيتم تنفيذ else. و هذه صورة توضح لطريقة عملهما:



الشكل ١,٧,٣: طريقة عمل if و else

١,٧,٣ استعمال else…if:

في المثال السابق لطريقة استعمال else إذا قمت بتنفيذ البرنامج و أعطيته القيمة ١٠٠ فسيقول لك أن النتيجة الـتي أدخلتها أكبر من ١٠٠، و لتفادي هذا المشكلة نستعمل الكلمتين else…if معا، و هذا مثال يوضح ذلك:

```
1   #include<stdio.h>
2
```

```
3   main()
4   {
5           int usr_val;
6
7           printf("Enter a value: ");
8           scanf("%d", &usr_val);
9
10          if(usr_val<100)
11                  printf("%d are less than 100\n", usr_val);
12          else if(usr_val==100)
13                  printf("%d are equal 100\n", usr_val);
14          else
15                  printf("%d are great than 100\n", usr_val);
16  }
```

البرنامج ١,٧,٤: طريقة إستعمال else…if

و طريقة عمل هذا المثال هو المقارنة بين القيمة التي أدخلها المستخدم و ١٠٠ فإذا كانت أقل من ١٠٠ فسنطبع ما هو موجود في *block* الخاص بالمقارنة، و في حالة أن القيمة أكبر من ١٠٠ فسيتم الانتقال إلى المقارنة الثانية و هي إذا كان العدد الذي أدخله المستخدم يساوي ١٠٠، في حالة أنها صحيح يقوم البرنامج بطباعة ما هو موجود في *block* الخاص بالمقارنة الثانية، في حالة أن كلا المقارنتين خاطئة فسيتم تنفيذ ما هو موجود في الـ *block* الخاص بـ else.

١,٧,٤ الأخطاء المحتملة:

١. لا يمكن استعمال else مرتين لمقارنة واحدة.

٢. لا يمكن استخدام else بدون if.

١,٧,٥ تمارين:

١. أكتب برنامج يقوم بالمقارنة بين عمر المستخدم و العدد ٤٠، إذا كان عمر المستخدم أقل من ٤٠ يخبره البرنامج بأن عمره أقل من الأربعين، و في حالة العكس يخبره البرنامج بأن عمره أكبر من ٤٠، و إذا كان عمره أربعين يخبره البرنامج بأن عمره ٤٠ سنة.

٢. أكتب نفس البرنامج السابقة و لكن بدون استعمال else.

٣. أكتب برنامج يطلب من المستخدم إدخال قيمة لا تتعدى العدد ٩، و عند إدخال أي عدد يخبره البرنامج أنه قد أدخل ذلك العدد (باستعمال else…if).



١,٨ عناصر لغة C

في هذا الدرس سنتعرف على عناصر لغة *C* و التي هي:

١,٨,١ التعليقات Comments:

هي من عناصر لغة *C* و قد درسنها سابقا. التعليقات هي سلسلة من الرموز تبدأ بالرمز *slash* / و النجمة * و تنتهي بنجمة و *slash* */، و يمكن أن يحمل التعليق أي رموز أو أرقام. الطريقة السابقة هي الطريقة الوحيدة في لغة *C*، و لكن الكثير من المترجمات تدعم طريقة أخرى من التعليق و هي التعليق السطري حيث يبدأ بالرمزين //.

١,٨,٢ الكلمات المحجوزة Keywords:

الكلمات المحجوزة هي كلمات أساسية في اللغة و التي يكون لونها في أغلب المترجمات أزرق، و سميت بالكلمات المحجوزة لأنها محجوزة سابقا، كل من `int`، `char`، `double`، `if` و `else` تعتبر كلمات محجوزة و الكثير منها، و هذا جدول لجميع الكلمات المحجوزة الأساسية في لغة *C*:

int	char	else	volatile	return	void	struct	float
short	signed	register	for	continue	typedef	case	static
long	unsigned	auto	while	break	union	switch	default
double	if	const	do	sizeof	enum	extern	goto

الجدول ١,٨,١: الكلمات المحجوزة للغة C

و الكلمة المحجوزة `asm`، عددها ٣٣، و سندرس جميع الكلمات المحجوزة في الدروس القادمة.

١,٨,٣ المعرفات Identifiers:

المعرفات هي سلسلة من حروف أو أرقام، بدايتها يجب أن تكون حرفا، و لا يمكن أن يبدأ اسم المعرف برقم، الرمز _ و الذي يدعى *underscore* يعتبر حرفا، و لا يمكن أن يتعدى اسم المعرف أكثر من ٣١ حرفا، كل من الأحرف الصغير و الأخر الكبيرة مختلفة، و لا يمكن استعمال معرف مرتين. جدول للأحرف التي يمكن أن تستعمل في أسماء المعرفات:

الأحرف	الأرقام
a b c d e f g h i j k l m n o p q r s t u v w x y z _ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	٠ ١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩

الجدول ١,٨,٢: حدود أسماء المعرفات

١,٨,٣,١ سلاسل Trigraphs:

هي سلاسل مكونة من ثلاثة رموز يقوم المترجم باستبدالها إلى رموز بديليه، و هذا جدول توضيحي لهذه الرموز:

Trigraphs	الرمز الذي يقابله
??=	#
??<	{
??>	}
??(	[
??)	]
??/	\
??'	^
??!	|
??-	~

الجدول ١,٨,٣: Trigraphs

و هذا مثال يوضح طريقة استعمال تلك الرموز:

```
1  ??=include<stdio.h>
2
3  main()
4  ??<
5        printf("Hello, World!\n");
6  ??>
```

البرنامج ١,٨,١: إستعمال رموز Trigraphs

و البرنامج يعمل بدون أخطاء، و يمكن أيضا استعمال تلك الرموز داخل النصوص مثل:

```
1  #include<stdio.h>
2
3  main()
4  {
5        printf("??=\n");
6        printf("??<\n");
7        printf("??>\n");
8        printf("??(\n");
9        printf("??)\n");
10       printf("??//\n");
11       printf("??'\n");
12       printf("??!\n");
13       printf("??-\n");
14 }
```

البرنامج ١,٨,٢: إستعمال رموز Trigraphs (٢)

١,٨,٤ الثوابت Constants:

الثوابت تكون إما أرقام أو أحرف أو سلاسل حرفية، لا يمكن التغير فيها أثناء البرنامج، و توجد نوعان منها و هي:

١,٨,٤,١ الثوابت النصية:

الثوابت النصية هي تكون إما بالاستعمال الموجه #define أو الكلمة المحجوزة const، بالنسبة للموجه #define فهذا مثال له:

```
1  #include<stdio.h>
2
3  #define Const_Str "Hello, World!\n"
4
5  main()
6  {
7        printf(Const_Str);
8
9        /*can't write Const_Str = "Hello, World2!\n"*/
10 }
```

البرنامج ١,٨,٣: الثوابت النصية

و أي نصوص أو أحرف معرفة باستخدام الموجه #define فهي ثابتة و لا يمكن التغير فيها. أما الثوابت باستخدام الكلمة المحجوزة const فسأعطي مثال لحرف ثابت، المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        const char ch = 'a';
6
7        printf("%c\n", ch);
8
9        /*you can't use this:
10       ch = 'b';
11       printf("%c\n", ch);
12       because it's a constant*/
13 }
```

البرنامج ١,٨,٤: ثابت حرفي

أما الثوابت النصية باستخدام الكلمة المحجوزة const فسندرسها فيما بعد. و توجد رموز ثابتة أخرى خاصة بلغة *C* و هي موضحة على الجدول التالي:

الرمز	التأثير	الرمز	التأثير
\a	الإنذار، تقوم بإصدار صوت من	\'	طباعة الرمز '

	اللوحة الأم، *Alert*		
\b	الفضاء الخلفي، *Backspace*	\"	طباعة الرمز "
\f	صفحة جديدة، *Form feed*	\?	طباعة الرمز ?
\n	سطر جديد، *New line*	\\	طباعة الرمز \
\r	العودة بمؤشر الكتابة إلى بداية السطر، *Carriage return*	\0oo	لأعداد نظام الثماني
\t	٨ فراغات في الاتجاه الأفقي	\xhh	لأعداد نظام السداسي عشر
\v	٨ فراغات في الاتجاه العمودي		

الجدول ١,٨,٤: ثوابت خاصة بلغة C

١,٨,٤,٢ الثوابت الرقمية:

هي الإعلان عن ثوابت بها قيم ثابتة لا يمكن التحديث فيها أثناء البرنامج، و يمكن أن تكون أي نوع من الأعداد short, long, int, unsigned, signed, float, double، و يمكن أيضا الإعلان عنها في كل من الموجه #define و الكلمة المحجوزة const، مثال حول الموجه #define:

```
1  #include<stdio.h>
2
3  #define Const_Num 5
4
5  main()
6  {
7        printf("%d\n", Const_Num);
8
9        /*can't write Const_Num = 6;*/
10 }
```

البرنامج ١,٨,٥: الثوابت الرقمية

عند الإعلان عن ثوابت باستخدام الموجه #define فإننا لا نقوم بتحديد نوعه، الموجه نفسه يقوم بتحديد نوع القيمة المدخلة. و هذا مثال باستخدام الكلمة المحجوزة const:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int  Const_Num = 5;
6        printf("%d\n", Const_Num);
7
8        /*can't write Const_Num = 6;*/
9  }
```

البرنامج ١,٨,٦: الثوابت الرقمية (٢)

١,٨,٥ الرموز Tokens:

هي ستة فئات و هي: المعرفات، الكلمات المحجوزة، الثوابت، السلاسل النصية، المؤثرات و الفواصل.

١,٨,٦ السلاسل النصية String literals:

السلاسل النصية أي الثوابت النصية، هي سلاسل من حروف محاطة باقتباسين "..."، ستفهم هذا الجزء من الدروس القادمة.

١,٨,٧ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

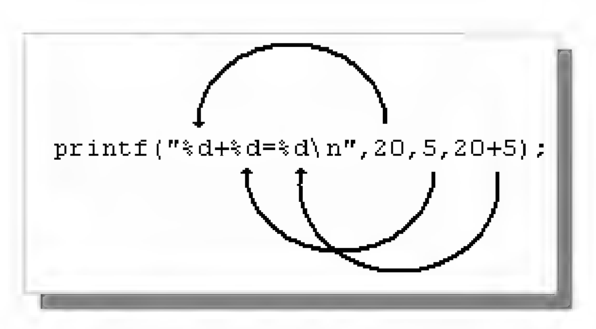
١,٨,٨ تمارين:

لا توجد تمارين في هذا الدرس.

١,٩ ملخص للفصل الأول، مع إضافات

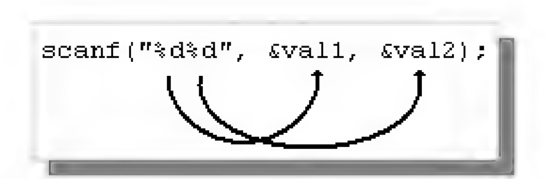
ما درسناه حتى الآن هو جزء شبه كبير من لغة C، و في هذا الملخص سنحاول تغطية و فهم أكثر لما درسناه.

درسنا من قبل الدالة printf، و قلنا أنها خاصة بالإخراج، و هنا توجد صورة توضيحية لطريقة تعامل هذه الدالة مع رموز مثل d%، الصورة:



الشكل ١,٩,١: طريقة عمل الدالة printf

من الصورة نفهم أنه عند تشغيل البرنامج يتم استبدال القيم المدخل أو القيم المعطاة في مكان الرمز d%، و أيضا نفس الفكرة مع الدالة scanf، صورة توضيحية:



الشكل ١,٩,٢: طريقة عمل الدالة scanf

١,٩,١ برامج تدريبية:

في هذا الجزء من الدرس سنرى بعض البرامج التي يستعملها كثيرا المبتدأين في لغة C، مع شرح سريع لكل برنامج:

١,٩,١,١ البرنامج الأول، عمر المستخدم:

يقوم هذا البرنامج بإعطاء المستخدم عمره، وذلك عبر إدخال العام الحالي و العام الذي ولد به المستخدم، المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int num1, num2;
6
7         printf("the year currently: ");
8         scanf("%d", &num1);
```

```
9
10        printf("the year of your born: ");
11        scanf("%d", &num2);
12
13        printf("You have %d years!\n", num1-num2);
14  }
```

البرنامج ١,٩,١: عمر المستخدم

تم الإعلان عن متغير في السطر الخامس، و في السطر السابع طلبنا من المستخدم إدخال السنة الحالية، و في السطر الثامن أخذنا السنة على شكل عدد حقيقي. في السطر العاشر طلبنا من المستخدم إدخال سنته، و في السطر الحادي عشر أخذنا السنة أيضا على شكل عدد صحيح. و أخيرا في السطر الثالث عشر قمنا بطباعة النتيجة و التي هي طرح السنة الحالية على سنة المستخدم.

١,٩,١,٢ البرنامج الثاني، آلة حاسبة بسيطة:

هذا البرنامج عبارة عن آلة حاسبة بسيطة ذات حدود لا يمكن تجاوزها، يطلب من المستخدم اختيار إشارة للعملية التي سيقوم بها، ثم إدخال قيمتين اللذان سيجري عليهما العملية، المثال:

```
1   #include<stdio.h>
2
3   main()
4   {
5         int num1, num2;
6         char Char;
7
8         printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9         scanf("%c", &Char);
10
11        printf("Enter the first number: ");
12        scanf ("%d", &num1);
13
14        printf("Enter the second number: ");
15        scanf("%d", &num2);
16
17        if(Char == '+' || Char == '1')
18              printf("%d + %d = %d\n", num1, num2, num1+num2);
19
20        else if(Char == '-' || Char == '2')
21              printf("%d - %d = %d\n", num1, num2, num1-num2);
22
23        else if(Char == '/' || Char == '3')
24              printf("%d / %d = %d\n", num1, num2, num1/num2);
25
26        else if(Char == '*' || Char == '4')
27              printf("%d * %d = %d\n", num1, num2, num1*num2);
28
29        else
30              printf("Error in choice!\nExiting...\n");
31  }
```

البرنامج ١,٩,٢: آلة حاسبة بسيطة

في السطر الخامس قمنا بالإعلان عن متغيرين من نوع أعداد صحيحة، و في السطر السادس قمنا بالإعلان عن متغير حرفي و الذي سيكون الإشارة التي سنقوم باستعمالها في العمليات. في السطر الثامن طلبنا من المستخدم بإختيار الإشارة أو المؤثر الذي يريد استعماله، و أخذنا المؤشر الذي إختاره المستخدم في السطر التاسع على شكل رمز. من السطر الحادي عشر إلى السطر الخامس عشر، قمنا بأخذ الأعداد التي أدخلها المستخدم. في السطر السابع عشر قمنا باستعمال شرط و هو إذا كان الرمز الذي أدخله المستخدم في المتغير Char يساوي المؤثر + أو الرقم ١، في حالة أن الشرط صحيح يتم تطبيق عملية الجمع، و هكذا بالنسبة لكل من الأسطر العشرين، الثالث و العشرين، السادس و العشرين، أما السطر التاسع و العشرين فسيتم تنفيذه إذا كان الرمز الذي أدخله المستخدم غير متوافق مع الرموز المطلوبة.

١,٩,١,٣ البرنامج الثالث، استخراج القيمة المطلقة:

هذا البرنامج يقوم باستخراج قيمة مطلقة لعدد يقوم بإدخاله المستخدم، المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int num, x;
6
7        printf("Enter a number: ");
8        scanf ("%d", &num);
9
10       if(num<0)
11             x = -num;
12       else
13             x = num;
14
15       printf("|%d|=%d\n", num, x);
16 }
```

البرنامج ١,٩,٣: استخراج القيمة المطلقة

في السطر الخامس قمنا بالإعلان عن متغيرين، الأول للعدد الذي سيقوم بإدخاله المستخدم، و المتغير الثاني هو الذي سيحمل القيمة المطلقة للعدد الذي أدخله المستخدم. في السطر السابع طلبنا من المستخدم إدخال عدد، و أخذنا العدد الذي أدخله المستخدم على شكل عدد صحيح في السطر الثامن. في السطر العاشر قمنا باستعمال شرط و هو إذا كان العدد الذي أدخله المستخدم أقل من الصفر أي العدد سالب فإن المتغير x يساوي قيمة المتغير num مع عكس الإشارة، و في حالة أن الشرط كان خاطئ فهذا يعني أن العدد الذي أدخله المستخدم عدد أكبر من الصفر أي عدد موجب. في السطر الخامس عشر قمنا بطباعة النتائج.

١,٩,١,٤ البرنامج الرابع، أخذ العدد الكبير:

يقوم هذا البرنامج بأخذ العدد الأكبر بين عددين يقوم بإدخالهما المستخدم، المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int num1, num2, max;
6
7        printf("Enter a number: ");
8        scanf("%d", &num1);
9
10       printf("Enter another number: ");
11       scanf("%d", &num2);
12
13       if(num1>num2)
14              max = num1;
15       else
16              max = num2;
17
18       printf("max = %d\n", max);
19 }
```

البرنامج ١,٩,٤: أخذ العدد الكبير

في السطر الخامس قمنا بالإعلان عن ثلاثة متغيرات، الأول و الثاني لأعداد التي سيقوم بإدخالها المستخدم، و المتغير الثالث لعدد الأكبر بين العددين الذي أدخلهما المستخدم. من السطر السابع إلى السطر الحادي عشر، طلبنا من المستخدم إدخال عددين. في السطر الثالث عشر قمنا باستعمال شرط و هو إذا كان العدد الأول الذي أدخله المستخدم أكبر من العدد الثاني فإن المتغير max يساوي قيمة المتغير num1 لأنها الأكبر، و في حالة أن الشرط خاطئ فهذا يعني أن العدد الثاني الذي أدخله المستخدم أكبر من الأول و هنا ستكون قيمة المتغير max هي قيمة المتغير num2 لأنه الأكبر. و أخيرا السطر الثامن عشر و هو يقوم بطباعة النتائج.

١,٩,٢ الدالتين putchar و getchar:

الآن سنرى دالتين جديدتين و هما putchar و هي مختصرة من *put character* و الدالة getchar و هي مختصرة من *get character*، الدالتين من دوال الملف الرأسي stdio.h. الدالة putchar تستعمل لإخراج الأحرف، مثلا لو أردنا أن نكتب الجملة *Hello, World !* سنكتب:

```
1  #include<stdio.h>
2
3  main()
4  {
5        putchar('H');
```

```
6          putchar('e');
7          putchar('l');
8          putchar('l');
9          putchar('o');
10         putchar(',');
11         putchar(' ');
12         putchar('w');
13         putchar('o');
14         putchar('r');
15         putchar('l');
16         putchar('d');
17         putchar('\n');
18  }
```

البرنامج ١,٩,٥: طريقة إستعمال الدالة putchar

و لا يمكن استعمالها لطباعة النصوص، و في هذه الحالة ستفضل استعمال printf. أما الدالة getchar فهي تأخذ من المستخدم حرف و تضعه في متغير، و هذا مثال:

```
1   #include<stdio.h>
2
3   main()
4   {
5          char ch;
6
7          putchar(':');
8
9          ch = getchar();
10
11         putchar(ch);
12  }
```

البرنامج ١,٩,٦: طريقة إستعمال الدالة getchar

و هنا كتبنا ch = getchar() لأن الدالة getchar لا تحتوي على وسائط، و هنا الطريقة صحيحة لأن الدالة getchar تقوم بإرجاع قيمة و هي الحرف الذي أدخله المستخدم.

١,٩,٣ الدالتين puts و gets:

الدالتين puts و gets أيضا من دوال الملف stdio.h، و الأولى مختصرة من *put string* و الثانية *get string*. الدالة puts مشابه لدالة printf مع بضع الاختلافات، و هي تستعمل لطباعة النصوص، و تختلف عن الدالة printf في:

١- يمكنك تجاهل وضع رمز السطر الجديد \n، لأن الدالة puts تضع كل نص في سطر.

٢- الدالة puts لا يمكنها التعامل مع متغيرات من نوع أرقام أو أحرف، أي أنها لا تتعامل مع رموز مثل %d.

و هذا مثال عن الدالة puts:

```
1  #include<stdio.h>
2
3  main()
4  {
5         puts("Hello, World!");
6         puts("Goodbay, World!");
7  }
```

البرنامج ١,٩,٧: طريقة إستعمال الدالة puts

أما عن الدالة gets فهي تقوم بإدخال نصوص، و لا يمكن التعامل معها بالأرقام و الأحرف، سنتطرق إليها في الدروس القادمة، لأننا لم نتعامل مع النصوص بعد.

١,٩,٤ الدالتين wprintf و wscanf:

هما نفس الدالتين printf و scanf و لكنها عريضة و هي مختصرة من *wide printf format* و *wide scan format*، هذا مثال يوضح طريقة استعمالهما:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int num, x;
6
7         wprintf(L"Enter a number: ");
8         wscanf(L"%d", &num);
9
10        if(num<0)
11               x = -num;
12        else
13               x = num;
14
15        wprintf(L"|%d|=%d\n", num, x);
16 }
```

البرنامج ١,٩,٨: طريقة إستعمال الدالة wprint و الدالة wscanf

و الحرف L يعني *long*.

١,٩,٥ الدالتين putch و getch و الدالة getche:

الدالتين putche و getche هما من الدوال القياسية في لغة *C*، في أنظمة *linux* تجد الـــدالتين في الملـــف الرأســـي stdio.h، أما في أنظمة *Windows* فستجدها في الملف الرأسي conio.h (مختصر من *Console Input Output*)، أمـــا

الدالة getch فهي ليس من الدوال القياسية للغة *C* و لكنها متوفر في جميع المترجمات تقريبا علـــى أنظمـــة *Windows*،و تستعمل بكثرة.

إذا كانت النسخة التي لديك من لغة *C* هي النسخة القياسية *ANSI C* فيمكنك الإعتماد على الملف الرأسي stdio.h، و في النسخة القياسية لا توجد الدالة getch. الدالة putch هي نفس الدالة putchar، و الدالة getch متشابها لـــدالة getchar مع بعض الاختلافات:

١- الدالة getchar تسمح لك برؤية ما قمت بإدخاله، أما getch فلا تسمح لك بذلك.

٢- يمكن إدخال أكثر من حرف في الدالة getchar حيث سيتم أخذ الحرف الأول من الحروف المدخلـــة، أمـــا getch فلا تستطيع إدخال أكثر من حرف.

مثال حول putch و getch:

```
1  #include<stdio.h>
2  #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4  main()
5  {
6        char ch;
7
8        ch = getch();
9        putch(c);
10       putch('\n');
11 }
```

البرنامج ١,٩,٩: طريقة إستعمال الدالة getch و الدالة putch

قمنا بضم الملف الرأسي conio.h في السطر الثاني، ثم استعملنا الدالة getch في السطر الثامن حيـــث ســـتلاحظ أن طريقة استعمالها مثل الدالة getchar، ثم استعملنا الدالة في كلا السطرين التاسع و العاشر، و بالنسبة لرمز \n فهو يعتبر رمز واحد. و في الكثير من البرامج ستجد أن الدالة getch مستعملة بكثرة و خاصة في نهاية البرنامج و ذلـــك لأنهـــا تتوقف بانتظار المستخدم بالضغط على زر من الأزرار و طريقة هي:

```
1  #include<stdio.h>
2  #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4  main()
5  {
6        char ch;
7
8        ch = getch();
9        putch(ch);
10       putch('\n');
```

```
11
12        printf("Press any key to exit\n");
13        getch(); /*pause */
14  }
```

البرنامج ١,٩,١٠: طريقة إستعمال الدالة getch و الدالة putch (٢)

حيث هنا لن يخرج البرنامج مباشرة، سينتظر أن يقوم المستخدم بالضغط على أي زر كي يقوم البرنامج بالخروج. أما الدالة getche فهي نفسها getch فقط عند استعمالها تظهر للمستخدم الرمز الذي قام بإدخاله، مثال:

```
1   #include<stdio.h>
2   #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4   main()
5   {
6         char c;
7
8         c = getche();
9         putch('\n');
10        putch(c);
11        putch('\n');
12  }
```

البرنامج ١,٩,١١: طريقة إستعمال الدالة getche

١,٩,٦ الكلمة المحجوزة wchar_t:

طريقة استعمالها مثل طريقة استعمال الكلمة المحجوزة char، حيث الكلمة wchar_t مختصرة من *wide-character type* و حجمها ٢ بايت، مثال لطريقة استعمالها:

```
1  #include<stdio.h>
2
3  main()
4  {
5        wchar_t wch = 'a';
6
7        printf("%c\n", wch);
8  }
```

البرنامج ١,٩,١٢: طريقة إستعمال الكلمة المحجوزة wchar_t

١,٩,٧ الدالة الرئيسية main و wmain:

الدالة main هي دالة رئيسية حيث يبدأ البرنامج بتنفيذ الأوامر منها، باتجاه واحد و بطريقة منظمة، الدالة wmain (ليست من الدوال القياسية للغة *C*) هي نفسها main و لكنها عريضة، و كل الدوال و الكلمات المحجوزة العريضة تستعمل في النظام الحروف الدولي الموحد أو بما يسمى بـ *Unicode*. و طرقة استعمال wmain مثل main و هذا مثال بسيط حولها:

```
#include<stdio.h>

wmain()
{
	wprintf(L"Hello, World!\n");
}
```

البرنامج ١,٩,١٣: الدالة wmain

في بعض المترجمات إن كتبت البرنامج التالي:

```
#include<stdio.h>

main()
{

}
```

البرنامج ١,٩,١٤: الدالة main

يحذرك بأن الدالة main يجب أن ترجع قيمة، و لإرجاع قيمة لدالة الرئيسية نكتب الكلمة المحجوزة return مع القيمة صفر و التي تعني الخروج من البرنامج بدون أخطاء، و هذا لطريقة استعمالها:

```
#include<stdio.h>

main()
{
	return 0;
}
```

البرنامج ١,٩,١٥: إرجاع قيمة لدالة الرئيسية

و سبب إرجاع قيمة لدالة الرئيسية هو الوضع الافتراضي لها و هو int، و جميع الدوال من نوع int يجب إرجاع قيمة لها، و هذا يعني أنه يمكن كتابة:

```
#include<stdio.h>

int main()
{
	return 0;
}
```

البرنامج ١,٩,١٦: إرجاع قيمة لدالة الرئيسية (٢)

أما إذا أردت عدم إرجاع قيمة لدالة الرئيسية فيمكن كتابة void بدل int، حيث تستعمل void للقيام بـــإجراءات و لا يمكن إرجاع لها قيمة، و هذا مثال:

```
#include<stdio.h>

```

```
3 void main()
4 {
5
6 }
```

البرنامج ١,٩,١٧: تفادي إرجاع قيمة لدالة الرئيسية

١,٩,٨ رموز الإخراج و الإدخال:

رموز الإخراج الخاصة بالدالة printf موضحة على الجدول التالي:

الرمز	الشرح
%d, %i	للأعداد الصحيحة و يمكن استعمالهما لكل من الأنواع int, short, long
%f, %e, %g	للأعداد الحقيقية و يمكن استعمالهم لكل من الأنواع float, double
%u	للأعداد بدون إشارة و يمكن استعماله لكل من الأنواع unsigned, int, short, long
%c	للرموز و الأحرف و يستعمل مع المتغيرات الحرفية char
%s	للنصوص أو السلاسل الحرفية و يمكن استعماله مع char[] و char*
%o	لأعداد النظام الثماني
%x	لأعداد النظام السداسي
%p	للمؤشرات
%ld	لأعداد صحيحة طويلة *Long Decimal* و تستعمل في long int
%lu	لأعداد طويلة بدون إشارة long unsigned

الجدول ١,٩,١: رموز الدالة printf

و رموز الإدخال الخاصة بالدالة scanf على الجدول التالي:

الرمز	الشرح
%d, %i	للأعداد الصحيحة و يمكن استعمالهما لكل من الأنواع int, short, long
%f, %e, %g	للأعداد الحقيقية و يمكن استعمالهم لكل من الأنواع float, double
%u	للأعداد بدون إشارة و يمكن استعماله لكل من الأنواع unsigned, int, short, long
%c	للرموز و الأحرف و يستعمل مع المتغيرات الحرفية char
%s	للنصوص أو السلاسل الحرفية و يمكن استعماله مع char[] و char*

%o	لأعداد النظام الثماني
%x	لأعداد النظام السداسي
%ld	لأعداد صحيحة طويلة *Long Decimal* و تستعمل في long int
%lu	لأعداد طويلة بدون إشارة long unsigned

الجدول ١,٩,٢: رموز الدالة scanf

١,٩,٩ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

١,٩,١٠ تمارين:

١. أكتب برنامج يطلب من المستخدم إدخال عدد، ثم يقوم بالبرنامج بإخباره إن كان العدد الذي أدخله فردي أو زوجي.

٢. أيهما أفضل في الإخراج الدالة printf أو الدالة puts؟ و لماذا؟

٣. ماذا تعني الكلمة conio، و ما هي الدوال التي درستها من الملف الرأسي conio.h ؟

٤. فيما تستعمل الكلمات المحجوزة و الدوال العريضة؟

٥. لماذا نقوم بإرجاع قيمة لدالة الرئيسية في حالة أنها في الوضع الافتراضي، و ماذا تعين تلك القيمة في الدالة الرئيسية؟

2

الفصل الثاني – أساسيات في لغة C (2)

مقدمة: في هذا الفصل سننتقل إلى المرحلة الثانية من لغة C، حيث سنتعرف إلى كيفية إستعمال القرار switch، طريقة تكرار أمر إلى عدة مرات (حلقات التكرار)، مصفوفات ذات بعد، بعدين، ثلاثة أبعاد و أكثر، الذاكرة و المؤشرات، الدوال و الإجراءات، الملفات الرأسية، طريقة التعامل مع الملفات سواء قراءة، كتابة، إنشاء و حذف، و أخيرا البرمجة التركيبية، و إنهاء هذا الفصل يعني إنهاء أساسيات لغة C.

بالتوفيق إن شاء الله

قال الله تعالى:

﴿ وقال الذين أوتوا العلم ويلكم ثواب الله خير لمن آمن وعمل صالحا ﴾

صدق الله تعالى

٢,١ القرار Switch

يستعمل القرار *switch* لإختبار قيمة متغيرة مع قيم ثابتة صحيحة، حيث كل قيمة ثابتة تعتبر كشرط أو مقارنة. و القرار *switch* مشابه للقرارات if، else و else…if، سيغنينا عن باقي هذه القرارات في الكثير من الحالات منها:

إليك هذا المثال الذي كتبناه سابقا و الذي هو عبارة عن آلة حاسبة بسيطة:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int num1, num2;
6         char Char;
7
8         printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9         scanf("%c", &Char);
10
11        printf("Enter the first number: ");
12        scanf ("%d", &num1);
13
14        printf("Enter the second number: ");
15        scanf("%d", &num2);
16
17        if(Char == '+' || Char == '1')
18              printf("%d + %d = %d\n", num1, num2, num1+num2);
19
20        else if(Char == '-' || Char == '2')
21              printf("%d - %d = %d\n", num1, num2, num1-num2);
22
23        else if(Char == '/' || Char == '3')
24              printf("%d / %d = %d\n", num1, num2, num1/num2);
25
26        else if(Char == '*' || Char == '4')
27              printf("%d * %d = %d\n", num1, num2, num1*num2);
28
29        else
30              printf("Error in choice!\nExiting...\n");
31 }
```

البرنامج ٢,١,١: آلة حاسبة بسيطة بإستخدام if, else, else…if

هنا يمكننا استعمال القرار switch أفضل، و هذا مثال باستخدامها:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int num1, num2;
6         char Char;
7
8         printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9         scanf("%c", &Char);
```

```
10
11         printf("Enter the first number: ");
12         scanf ("%d", &num1);
13
14         printf("Enter the second number: ");
15         scanf("%d", &num2);
16
17         switch(Char)
18         {
19         case '+':
20         case '1':
21                 printf("%d + %d = %d\n", num1, num2, num1+num2);
22                 break;
23
24         case '-':
25         case '2':
26                 printf("%d - %d = %d\n", num1, num2, num1-num2);
27                 break;
28
29         case '/':
30         case '3':
31                 printf("%d / %d = %d\n", num1, num2, num1/num2);
32                 break;
33         case '*':
34         case '4':
35                 printf("%d * %d = %d\n", num1, num2, num1*num2);
36                 break;
37
38         default:
39                 printf("Error in choice!\nExiting...\n");
40                 break;
41         }
42
43 }
```

البرنامج ٢,١,٢: آلة حاسبة بسيطة بإستخدام switch

هنا سنلاحظ أن عدد أسطر البرنامج أكثر من السابق، و هذا لا يهم، الذي يهم هو طريقة كتابة البرنامج حيث تكـون منظمة و مفهومة. للقرار switch وسيط، و الذي يقوم بتتبع نتيجة المتغير الموجود داخل الوسيط، في مثالنـا السـابقة استعمالنا المتغير Char، و للقرار switch حالات، و يتم تنفيذها حسب نتيجة المتغير الموجود في الوسيط. الآن سأشرح المثال السابق بالتفصيل:

في السطر الثامن طلبنا من المستخدم إدخال نوع العملة التي يريد استخدامها، و في السطر التاسع يأخذ البرنامج نوع العملية، و في السطر الحادي عشر و الرابع عشر طلبنا من المستخدم إدخال العدد الأول ثم العدد الثاني لإجراء العمليـة بينهما، و في السطر السابع عشر يأتي القرار switch و به وسيط في داخله المتغير Char حيث هنا ستتم المقارنة بـين القيمة الموجد داخل المتغير Char و الحالات الموجودة، في السطر التاسع عشر و العشرين سيقوم البرنامج بالمقارنة بـين الإشارة + و الرقم ١ بالمتغير Char فإذا كانت مقارنة واحدة صحيحة فسيتم تنفيذ السطر الواحد و العشرين قم الخروج من القرار switch في السطر الثاني و العشرين بدون متابعة المقارنة مع الحالات الأخرى، و في حالـة أن المقارنـة أو

الشرط لم يتحقق فسيتم الانتقال إلى الحالة التي بعدها و تأتي المقارنة معها و معا المتغير Char و هكذا...، و في السطر الثامن و ثلاثين تجد الكلمة المحجوزة default و سيتم تنفيذ ما هو بعدها إن لم يتحقق أي شرط من الشروط السابقة.

سأجعل المثال السابق مفهوم أكثر، لذا سأقوم بالمقارنة بينه و بين المثال الذي قبله في المقارنة الأولى و الأخيرة.

```
19        case '+':
20        case '1':
21              printf("%d + %d = %d\n", num1, num2, num1+num2);
22              break;
```

هذا في القرار switch، أما في القرارات if, else, else…if فهو:

```
17        if(Char == '+' || Char == '1')
18              printf("%d + %d = %d\n", num1, num2, num1+num2);
```

و هنا ستلاحظ أن السطرين:

```
19        case '+':
20        case '1':
```

هو نفسه:

```
Char == '+' || Char == '1'
```

أما الكلمة المحجوزة default:

```
38        default:
39              printf("Error in choice!\nExiting...\n");
```

فهي else في المثال السابق:

```
29        else
30              printf("Error in choice!\nExiting...\n");
```

و في القرار switch لا يمكن استعمال متغير من نوع أعداد حقيقية مثل float و double(هنا ستفضل if و else).

٢,١,١ الكلمة المحجوزة case:

هي من الكلمات المحجوزة و التي تعلمنها مؤخر، و هي لا تستعمل إلا في القرار switch، و هي تعني كلمة حالة، بعد اسم الحالة يأتي الشرط، فمثلا '1' case هنا سيتم المقارنة بين الرقم ١ و المتغير الموجود في وسيط القرار switch، فإذا كانت المقارنة صحيحة فسيتم تنفيذ ما هو بعد الحالة من أوامر، و في حالة أنها خاطئة فسيتم الانتقال إلى الحالات الأخرى. لا يمكن استعمال نص للمقارنة في الكلمة المحجوزة case، لأنها تتعامل مع الأرقام و الأحرف فقط، حتى القرار switch لا يمكن إعطائه متغير منت نوع سلسلة حروف، هو أيضا لا يقبل إلا بمغيرات الأحرف و الأرقام.

٢,١,٢ الكلمة المحجوزة break:

يمكن استعمالها داخل القرار switch و هي تعني الانقطاع، و هي تستعمل مع الكلمة المحجوزة case، في حالة الاستغناء عنها فستأتي نتائج غير مرغوب بها، و عملها هو الخروج من الحلقة القرار switch.

٢,١,٣ الكلمة المجوزة default:

أيضا تستعمل مع القرار switch و هي الوضع الافتراضي، أي أنه إن لم تتحقق أي حالة من الحالات السابقة فسيتم تنفيذ ما هو بعدها.

٢,١,٤ الأخطاء المحتملة:

١. في حالة أنك أردت استعمال الحالة مباشرة باستخدام الرقم بعد case فيجب عليك استعمال متغير من نوع أعداد صحيحة، و ليس متغير أحرف و المثال السابق سيصبح على هذا الشكل:

```
1  #include<stdio.h>
2
3  main()
4  {
5  	int num1, num2;
6  	int Char;
7
8  	printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9  	scanf("%d", &Char);
10
11 	printf("Enter the first number: ");
12 	scanf ("%d", &num1);
13
14 	printf("Enter the second number: ");
15 	scanf("%d", &num2);
16
17 	switch(Char)
18 	{
19 	case 1:
20 		printf("%d + %d = %d\n", num1, num2, num1+num2);
```

```
21                break;
22
23          case 2:
24                printf("%d - %d = %d\n", num1, num2, num1-num2);
25                break;
26
27          case 3:
28                printf("%d / %d = %d\n", num1, num2, num1/num2);
29                break;
30
31          case 4:
32                printf("%d * %d = %d\n", num1, num2, num1*num2);
33                break;
34
35          default:
36                printf("Error in choice!\nExiting...\n");
37          }
38
39    }
```

البرنامج ٢,١,٣: الخطأ ١

٢. يجب دائما الانتباه إلى الكلمة المحجوزة `break` و مكان استعمالها.

٢,١,٥ تمارين:

١- أكتب برنامج به قائمة رئيسية للعبة، حيث سيكون لها أربع خيارات و هي:

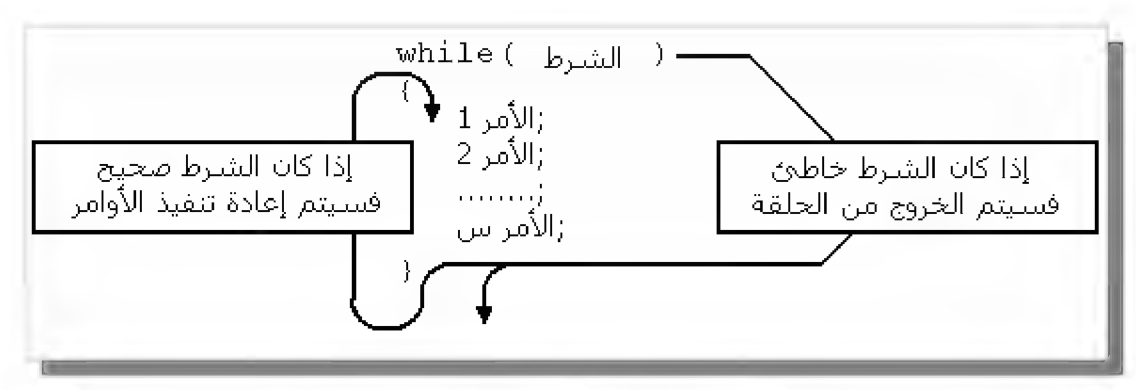
الخيار الأول هو *Start Game* و عند استعمال هذا الخيار مخبره بأن اللعبة قد بدأ. و الخيار الثاني هو *Option* و التي به هي أيضا أربع خيارات لكل من تعديل الصوت و الصورة و الفأرة و لوحة التحكم. الخيار الثالث هو *About Game* و التي ستضع فيها معلومات عنك. و الخيار الأخير هو الخروج من اللعبة *Exit Game*.

٢,٢ حلقات التكرار Repeated loops

معنى حلقات التكرار بصفة عامة هي تكرار شيء، و في البرمجة التكرار يكون لحلقة يتم تكرارها لمـــرات يقـــوم بتحديدها المبرمج، توجد ثلاثة طرقة أساسية في التكرار، و توجد طريقة رابعة و لكنها لا تعتبر من طرق التكرار، الطرق هي:

٢,٢,١ التكرار بواسطة while:

طريقة استعمالها مشابه قليلا لطريقة استعمال الكلمة المحجوزة if، و هذه صورة بها شرح لطريقة استعمالها:



الشكل ٢,٢,١: التكرار بواسطة while

سيتم تنفيذ الأوامر بدون توقف حتى يصبح الشرط خاطئ، و إن كان شرط ثابت أو أن قيمته لا تتغير فـــسيتم تنفيـــذ الأوامر مرات لا حدود لها، أي سيبقى تنفيذ الأوامر بدون توقف لذا يجب أن نستعمل مؤثرات الزيادة أو النقـــصان أو نعطي الحرية للمستخدم بإيقاف الحلقة متى أراد. الآن سأعطي مثال بسيط حول طريقة استعمال التكرار بواسطة الكلمة المحجوزة while حيث يقوم بطباعة عد تصاعدي من ١ إلى ٣:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int i;
6        i = 1;
7
8        while(i!=4)
9        {
10             printf("\a%d ,", i);
11             i++;
12       }
13
14       printf(" Go!\n");
15 }
```

البرنامج ٢,٢,١: التكرار بواسطة while

في حلقات التكرار دائما نستعمل متغيرات و إلى ستكون الحلقة بلا نهاية، و هنا استعملنا المتغير i و الذي سيكون العداد للحلقة، و يجب أيضا إعطاء للمتغير قيمة يبدأ بها، في مثالنا أعطيناه القيمة ١ حيث سيبدأ العد من الرقم ١، و في السطر الثامن وضعنا الكلمة المحجوزة while مع شرط (مقارنة) أن لا يكون المتغير i يساوي ٤ حيث سيتم تنفيذ ما هو داخل الـ *block* الخاص بـ while إلى أن تصبح المقارنة خاطئة (حيث سيتم الخروج من الحلقة إن كان المتغير i يساوي ٤ و هذا يعني أن الشرط i != 4 سيكون خاطئ)، و ستجد أيضا أنه استعملنا مؤثر الزيادة في السطر الحادي عشر، و عدم استعماله يعطينا نتائج ثابت و غير منتهية. المثال السابق لمتغير من نوع عدد صحيح، الآن سأعطي مثال بسيط لطريقة استعمال التكرار لمتغير حرفي:

```
1  #include<stdio.h>
2  #include<conio.h>   // Just in Windows and DOS OS compilers
3
4  main()
5  {
6         char ch;
7
8         printf("Press any key(q=exit): ");
9         ch = getche();
10
11        while(ch != 'q')
12        {
13               printf("\nPress any key(q=exit): ");
14               ch = getche();
15        }
16
17        printf("\nExiting...\n");
18 }
```

البرنامج ٢,٢,٢: التكرار بواسطة while (٢)

استعملنا متغير حرفي في السطر السادس ثم طلبنا من المستخدم إدخال أي حرف، و استعمالنا الدالة getche لكي لا نسمح للمستخدم بإدخال أكثر من حرف، و في السطر الحادي عشر استعمالنا الكلمة المحجوزة while مع شرط أن لا يكون المتغير ch يساوي الحرف q، حيث هنا سيبقى البرنامج يعمل بلا نهاية إلا إذا ضغط المستخدم على الحرف q.

يمكننا الاستغناء عن الشرط في الكلمة المحجوزة while و كتابته داخل الـ *block* الخاص بها باستعمال المقارنة عبر الكلمة المحجوزة if حيث سيكون داخل الـ *block* الخاص بها الكلمة المحجوزة break و التي قلنا عليها سابقا أنها تنهي الحلقات، هذا المثال السابق بعد التعديل:

```
1  #include<stdio.h>
2  #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4  main()
```

```
5  {
6        char ch;
7
8        printf("Press any key(q=exit): ");
9        ch = getche();
10
11       while(1)
12       {
13             if(ch == 'q')
14                   break;
15
16             printf("\nPress any key(q=exit): ");
17             ch = getche();
18       }
19
20       printf("\nExiting!\n");
21 }
```

البرنامج ٢,٢,٣: التكرار بواسطة while (٣)

و هنا استعملنا الرقم ١(و الذي يعني true في لغة *C*) في وسيط الكلمة المحجوزة while كي تكون الحلقة بلا نهاية، و هنا ستلاحظ أن الكلمة المحجوزة break لا تستعمل فقط في الكلمة المحجوزة switch. في المثال السابق يمكننا الاستغناء عن السطر الثامن و التاسع و نجعل الشرط الموجود في السطر الثالث عشر في آخر الحلقة حيث سيصبح المثال كما يلي:

```
1  #include<stdio.h>
2  #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4  main()
5  {
6        char ch;
7
8        while(1)
9        {
10             printf("\nPress any key(q=exit): ");
11             ch = getche();
12
13             if(ch == 'q')
14                   break;
15       }
16
17       printf("\nExiting!\n");
18 }
```

البرنامج ٢,٢,٤: التكرار بواسطة while (٤)

في هذا المثال سيتم تنفيذ الأوامر الموجود في الـ *block* الخاص بـ while مرة واحدة على الأقل، حتى و إن تحقق الشرط، و لكنها ليست من ميزات الحلقة while، حيث تستعمل هذه طريقة غالبا أو ربما لا تستعمل أصلا.

٢,٢,٢ التكرار بواسطة do...while:

طريقة استعمالها مثل الطريقة السابقة لـ while مع بعض الإضافات، و الفرق بين do...while و while هو أن التكرار باستخدام while لا يقوم بتنفيذ الأوامر الموجود فيه و لا مرة واحدة إلا إذا كانت الشرط صحيح مرة واحدة، أما التكرار بواسطة do...while فسيتم تنفيذ الأوامر الموجود بالـ *block* الخاص به مرة واحدة على الأقل حتى و إن كان الشرط خاطئ. طريقة استعمالها هي كتابة الكلمة المحجوزة do ثم نقوم بوضع *block* خاص بها حيث ستكون بـه الأوامر المراد تنفيذها، و في رمز نهاية الـ block و الذي هو } نكتب while مع الشرط، و هذه صورة توضيحية:

```
do
{
    الأمر 1;
    الأمر 2;
    ........;
    الأمر س;
}
while (المقارنة)
```

إذا كان الشرط خاطئ فسيتم الخروج من الحلقة

إذا كان الشرط صحيح فسيتم إعادة تنفيذ الأوامر

الشكل ٢,٢,٢: التكرار بواسطة do...while

و هذه الأمثلة السابق باستخدام do...while:

مثال العد التصاعدي:

```
1   #include<stdio.h>
2   #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4   main()
5   {
6           int i;
7           i = 1;
8
9           do
10          {
11                  printf("\a%d ,", i);
12                  i++;
13          }while(i != 4);
14
15          printf(" Go!\n");
16  }
```

البرنامج ٢,٢,٥: التكرار بواسطة do...while

و هذا مثال المتغير الحرفي:

```
1  #include<stdio.h>
2  #include<conio.h>    /* Just in Windows and DOS OS compilers */
3
4  main()
5  {
6          char ch;
7
8          do
9          {
10                 printf("\nPress any key(q=exit): ");
11                 ch = getche();
12         }while(ch != 'q');
13
14         printf("\nExiting...\n");
15 }
```

البرنامج ٢,٢,٦: التكرار بواسطة do...while (٢)

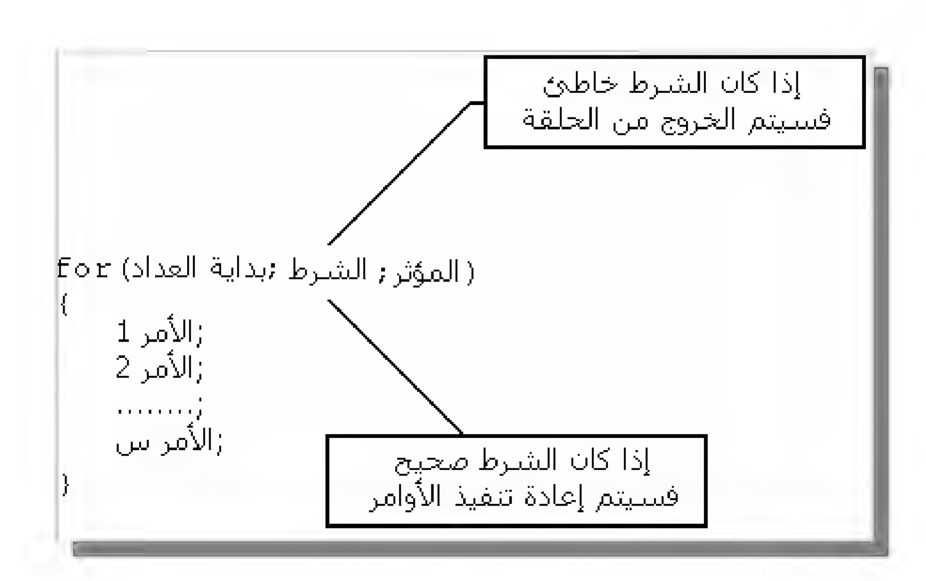
هنا ستجد أن عدد الأسطر قد قل، و لكن لا توجد طريقة أحسن من الأخرى لأن كل طريقة و مكان استعمالها.

٢,٢,٣ التكرار بواسطة for:

تعتبر هذه الطريقة من أسهل الطرق السابقة في الاستعمال، هي الطريقة الأكثر اسـتعمالا لبـساطتها و سـهولتها. للكلمة المحجوزة for ثلاثة وسائط و يتم الفصل بينها بفاصلة منقوطة بحيث:

١. الوسيط الأول نضع به قيمة التي سيبدأ بها العداد.
٢. الوسيط الثاني هو الشرط، في حالة أن الشرط كان خاطئ يتم الخروج من الحلقة.
٣. الوسيط الثالث هو المؤثر (غالبا ما يكون مؤثر الزيادة أو مؤثر النقصان).

صورة توضيحية:



الشكل ٢,٢,٣: التكرار بواسطة for

مثال العد التصاعدي سيكون على هذه الطريقة:

```
1  #include<stdio.h>
2  #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4  main()
5  {
6        int i;
7
8        for(i=1;i<=3;i++)
9        {
10             printf("\a%d ,", i);
11       }
12
13       printf(" Go!\n");
14 }
```

البرنامج ٢,٢,٧: التكرار بواسطة for

هنا نقوم أولا بالإعلان عن المتغير في بداية البرنامج ثم نعطي المتغير قيمة داخل الكلمة المحجوزة for أي في أول وسيط لها حيث ستكون تلك القيمة هي بداية العداد، ثم المقارنة في الوسيط الثاني حيث يجب أن لا يتعدى المتغير i العدد ٣ (إذا تعدى المتغير i العدد ٣ فسيصبح الشرط خاطئ و سيتم الخروج من الحلقة)، و في الوسيط الثالث استعملنا مؤثر الزيادة.

أما بالنسبة لمثال المتغير الحرفي فالطريقة تختلف قليلا، حيث سنقوم بالاستغناء عن الوسيط الأول و الأخير، و سنستعمل الوسيط الثاني لوضع الشرط حيث إن كان الشرط خاطئ يتم الخروج من الحلقة، و الشرط هنا هو أن لا يساوي المتغير ch الحرف q (سيصبح الشرط خاطئ إذا كان المتغير ch يساوي الحرف q)، و البرنامج سيكون على هذا الشكل:

```
1  #include<stdio.h>
2  #include<conio.h>   /* Just in Windows and DOS OS compilers */
3
4  main()
5  {
6        char ch;
7
8        printf("Press any key(q=exit): ");
9        ch = getche();
10
11       for(;ch!='q';)
12       {
13             printf("\nPress any key(q=exit): ");
14             ch = getche();
15       }
16
```

```
17          printf("\nExiting...\n");
18  }
```

البرنامج ٢,٢,٨: التكرار بواسطة for (٢)

في السطر الحادي عشر استعملنا التكرار مع تجاهل الوسيطين الأول و الأخير و لكن يجب أن نترك مكانا لهما، و استعملنا الوسيط الثاني و هو الشرط الذي في حالة كان خاطئ يتم الخروج من الحلقة.

٢,٢,٤ التكرار بواسطة goto:

هذه الطريقة إضافية، الكلمة المحجوزة goto تستعمل لـ التنقل من مكان لآخر في مصدر البرنامج، سأشرح أولا طريقة استعمالها ثم نذهب إلى التكرار بواسطتها. طريقة استعمالها بسيطة جدا، نكتب الكلمة المحجوزة goto ثم اسم المكان الذي نريد الذهاب إليه، هذه صورة توضيحية:

```
goto the_name_of_place;
```

الشكل ٢,٢,٤: طريقة الذهاب إلى مكان ما في البرنامج عبر goto

و لكن هذا الكود لا يكفي، يجب أن نضع اسم المكان حيث سيكون على الشكل التالي:

```
the_name_of_place:
```

الشكل ٢,٢,٥: إنشاء إسم لمكان يمكن الذهاب إليه عبر goto

الآن سأضع مثال بسيط مفهوم:

```
1   #include<stdio.h>
2
3   main()
4   {
5          goto fin;
6          printf("Begin!\n");
7
8   fin:
9          printf("End!\n");
10  }
```

البرنامج ٢,٢,٩: طريقة إستعمال goto

هنا عند تنفيذ البرنامج سيتم تجاهل كل ما بين the_name_of_place و goto و اسم المكان، و هنا اسم المكان الذي نريد الانتقال إليه هو fin و هو موجود في السطر الثامن، و أردنا الذهاب إلى المكان fin في بداية البرنامج. إذا أردنا الرجوع إلى بداية البرنامج فالطريقة سهلة، نكتب كما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
5  begin:
6          goto fin;
7          printf("Begin!\n");
8
9  fin:
10         printf("End!\n");
11         goto begin;
12 }
```

البرنامج ٢,٢,١٠: طريقة إستعمال goto (٢)

هنا ستجد تأثر التكرار، حيث هنا حلقتنا لن تنتهي، و إذا أردنا أن نجعل البرنامج يقوم بتكرار نحدده نحن فسنقوم باستعمال متغير و الذي سيكون العداد ثم مؤثر ثم الشرط، و سيصبح المثال على هذا الشكل:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int i;
6          i = 0;
7
8          printf("Begin!\n");
9
10 begin:
11         printf("%d ,", i);
12         i++;
13         if(i==10)
14                 goto fin;
15         else
16                 goto begin;
17
18 fin:
19     printf("\nEnd!\n");
20 }
```

البرنامج ٢,٢,١١: التكرار بواسطة goto

و هنا نكون قد وضعنا تكرار بواسطة الكلمة المحجوزة goto.

٢,٢,٥ المفهوم العام لحلقات التكرار:

لكي تفهم التكرار بصفة عامة فكل ما عليك معرفته هو:

١. لكل حلقة تكرار بداية و التي تسمى ببداية العداد، هنا توجد حالات يمكن فيها تجاهل وضع بداية التكرار، مثل استعمال متغيرات حرفية.

٢. لكل حلقة تكرار شرط، في حالة كان خاطئ يتم الخروج من الحلقة، حيث تجاهل وضع الشرط ينتج حلقة غير منتهية.

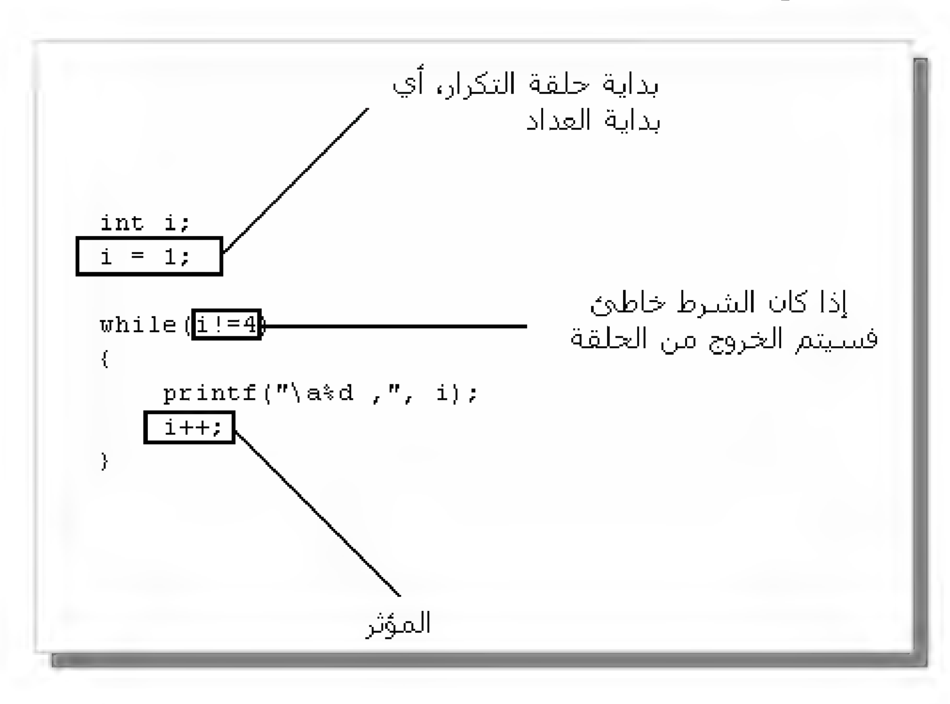
٣. لكل حلقة مؤثر سواء مؤثر الزيادة أو النقصان، هنا أيضا توجد حالات يمكن فيها تجاهل وضع مـؤثر، مثـل استعمال متغيرات حرفية.

مثالنا الأول في هذا الدرس هو:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int i;
6        i = 1;
7
8        while(i!=4)
9        {
10             printf("\a%d ,", i);
11             i++;
12       }
13
14       printf(" Go!\n");
15 }
```

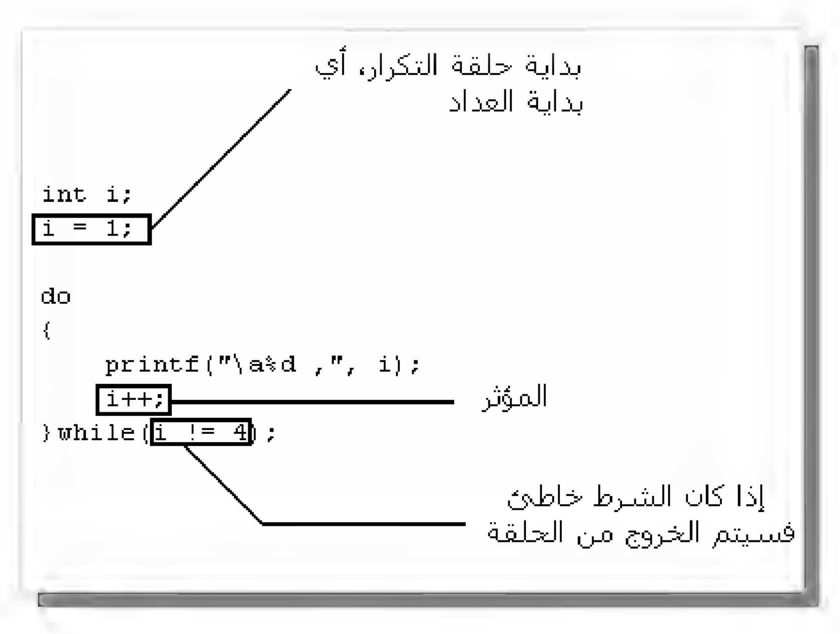
البرنامج ٢,٢,١٢: التكرار بواسطة while (٥)

و شرحه بصفة عامة موجود على الصورة التالية:



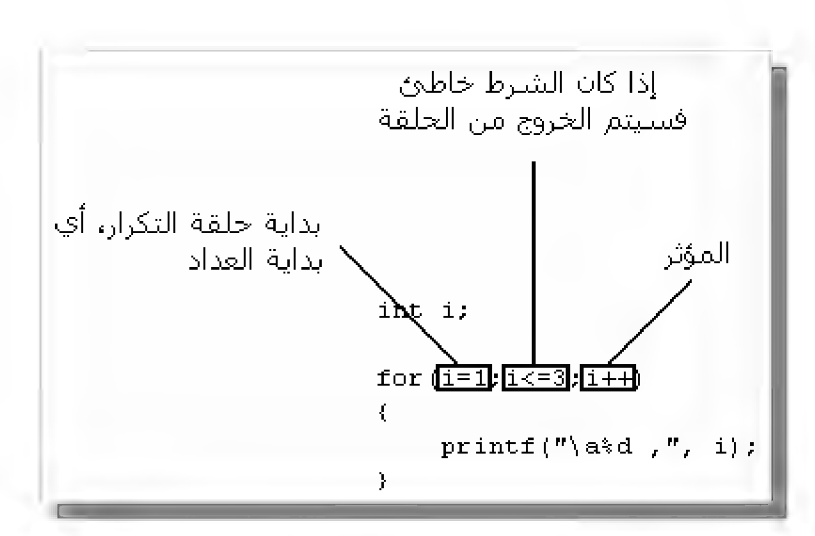
الشكل ٢,٢,٦: شرح لعملية التكرار في while

هذه الصورة لتكرار بواسطة while، و هذه صورة أخرى عن التكرار بواسطة do...while:



الشكل ٢,٢,٧: شرح لعملية التكرار في do...while

و أخيرا صورة توضيحية للتكرار بواسطة for:



الشكل ٢,٢,٨: شرح لعملية التكرار في for

و تبقى طريقة التكرار goto، تلك الطريقة يمكننا تجاهلها لأنها طريقة إضافية و هي غالبا ما تستعمل في التكرار. جميع الطرق السابقة يمكن استعمالها بطريقة أخرى، حيث يمكن أن نقوم بكتابة كل من بداية العداد و الشرط و المؤثر خارج وسائط كل من while, do...while, for، و هذا مثل طبيعي:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int i;
6
7          for(i=1;i<=3;i++)
8          {
9                  printf("\a%d ,", i);
10         }
11
12         printf(" Go!\n");
13 }
```

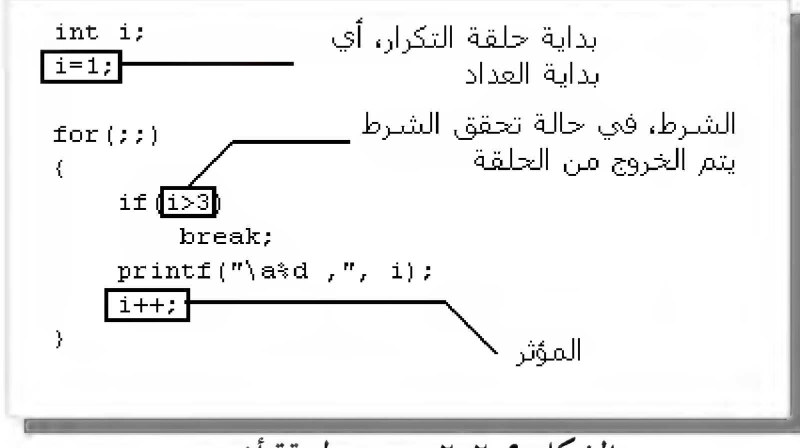
البرنامج ٢,٢,١٣: التكرار بواسطة for (٣)

و هنا نفس المثال السابق و لكن بطريقة أخرى:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int i;
6          i=1;
7
8          for(;;)
9          {
10                 if(i>3)
11                         break;
12                 printf("\a%d ,", i);
13                 i++;
14         }
15
16         printf(" Go!\n");
17 }
```

البرنامج ٢,٢,١٤: التكرار بواسطة for (٤)

و لكي تفهما هذا المثال إليك صورة توضحه:



الشكل ٢,٢,٩: for بطريقة أخرى

و هنا ستجد أن النتيجة ستكون نفسها، و المثال هنا باستخدام for، يمكن استعمال while و do...while، بنفس الفكرة.

٢,٢,٧ الكلمة المحجوزة continue:

تستعمل الكلمة المحجوزة continue داخل حلقات التكرار، حيث تقوم بالرجوع إلى بداية الحلقة بدون إكمال ماهو بعدها من أوامر، و كي تعرف أهميتها جرب المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int i;
6         i = 1;
7
8         while(1)
9         {
10               if(i<=3)
11               {
12                      printf("\a%d ,", i);
13                      ++i;
14               }
15
16               printf(" Go!\n");
17               break;
18        }
19
20 }
```

البرنامج ٢,٢,١٥: التكرار بواسيطة while (٦)

هذا المثال لن يعمل كما ينبغي، نريد من البرنامج أن يقوم بعد تصاعدي من ١ إلى ٣ أما هذا البرنامج لن يطبع إلى العدد ١، و كي يعمل البرنامج بطريقة صحيحة نظيف الكلمة المحجوزة continue إلى في نهاية المقارنة و سيصبح المثال السابق كما يلي:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int i;
6         i = 1;
7
8         while(1)
9         {
10               if(i<=3)
11               {
12                      printf("\a%d ,", i);
13                      ++i;
```

```
14                      continue;
15              }
16
17              printf(" Go!\n");
18              break;
19          }
20
21  }
```

البرنامج ٢,٢,١٦: الكلمة المحجوزة continue

و يمكن استعمال المثال السابق على كل من do...while و for.

٢,٢,٨ جدول ASCII:

باستخدام حلقات التكرار يمكننا معرفة كل رمز في جدول *ASCII* و رقمه، و ذلك باستخدام المثال التالي:

```
1   #include<stdio.h>
2
3   main()
4   {
5       int i;
6
7       for(i=0;i<=255;i++)
8           printf("%d: %c\n", i, i);
9
10  }
```

البرنامج ٢,٢,١٧: طباعة جدول ASCII بإستخدام حلقات التكرار

و ستجد عند تشغيل البرنامج أن بعض الرموز لم تطبع منها الرقم ٠ و الرقم ٧، ٨، ٩، ١٣ و أخرى، و سبب في ذلك أنها لا تقوم بأعمال ظاهرة فمثلا الرقم ٠ هو الرمز \0 و الرقم ١٣ هو زر الدخول في لوحة المفاتيح(أي سطر جديد).

٢,٢,٩ الأخطاء المحتملة:

١. يجب دائما الانتباه إلى الشرط و طريقة استعمالها.

٢. تذكر دائما أن التكرار بواسطة do...while يتم تنفيذ أوامره مرة واحدة على الأقل، حتى و إن كان الشرط خاطئ.

٣. استعمال حلقة التكرار for فارغة الوسائط تنتج حلقة غير منتهية، حيث ستكون على الشكل for(;;).

٤. استعمال الطريقة while(1) يعني حلقة بلا نهاية.

٥. استعمال الكلمة المحجوزة continue بدون شرط(أو تكرار بدون شرط) يعني حلقة بلا نهاية.

٢,٢,١٠ تمارين:

١. أكتب برنامج يقوم بالعد التنازلي من ١٠٠ إلى ٠، بالطرق الثلاثة.

٢. أكتب برنامج يقوم بطباعة الأعداد الثنائية من ٠ إلى ١٠٠، بالطرق الثلاثة.

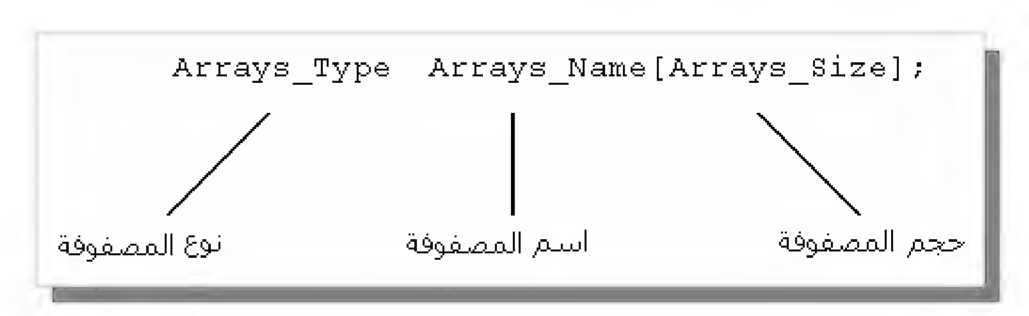
٢,٣ المصفوفات Arrays

تستعمل المصفوفات لإدارة مجموعة كبيرة من البيانات لها نفس النوع، و باستخدام اسم واحد، و يمكن أن تكون المصفوفة من أي نوع من أنواع المتغيرات، و لا يمكن استعمالها معا الدوال. فائدة المصفوفات كبيرة، و طرق استعمالها كثيرة و متنوعة، و لكي ترى فائدتها بشكل كبير فتخيل أنه طلب منك بناء برنامج به أكثر من ٢٠ متغير و كل متغير به قيمة ربما نقوم بتحديدها نحن أو يقوم بتحديدها المستخدم، و إليك المثال:

```
1   #include<stdio.h>
2
3   main()
4   {
5         int arr1, arr2, arr3, arr4, arr5, ...,arr25;
6
7         printf("Arr1: ");
8         scanf("%d", arr1);
9
10        printf("Arr2: ");
11        scanf("%d", arr2);
12
13        printf("Arr3: ");
14        scanf("%d", arr3);
15
16        printf("Arr4: ");
17        scanf("%d", arr4);
18
19        printf("Arr5: ");
20        scanf("%d", arr5);
..
..        ...
..
..        printf("Arr25: ");
..        scanf("%d", arr25);
..
..  }
```

البرنامج ٢,٣,١: برنامج به أكثر من ٢٠ متغير

تخيل...كم سيكون عدد أسطر البرنامج إن قمنا بكتابة جميع المتغيرات و ترك المستخدم يعطي لها قيم، ثم ماذا لو أردنا طبع النتائج، سيكون عدد أسطر البرنامج أكثر من ٢٠٠ سطر و سيكون البرنامج جامد، غير مرن و غير مفهوم. هنا تكمن فائدة المصفوفات، طريقة استعملها و التعامل مها مشابه لطريقة التعامل مع المتغيرات، كل ما ستفعله هو كتابة نوع المصفوفة ثم اسمها ثم يأتي حجمها و هذه صورة توضيحية:



الشكل ٢,٣,١: طريقة الإعلان عن مصفوفة

و الآن سأكتب المثال السابق باستخدام المصفوفات:

```
1  #include<stdio.h>
2
3  main()
4  {
5      int arr[24];
6      int i;
7
8      for(i=0;i<25;i++)
9      {
10         printf("Arr%d: ", i);
11         scanf("%d", &arr[i]);
12     }
13
14     printf("***********- LIST -***********\n");
15
16     for(i=0;i<25;i++)
17     {
18         printf("Arr%d: %d\n", i, arr[i]);
19     }
20
21 }
```

البرنامج ٢,٣,٢: برنامج به أكثر من ٢٠ متغير بإستخدام المصفوفات

الآن أصبح البرنامج أكثر مرونة و مفهوم مع قلة عدد الأسطر، إن كان هذا المثال صعب فيمكن الرجوع إليه بعدما تقرأ هذا الدرس جيدا.

٢,٣,١ أساسيات في المصفوفات:

إذا كانت لدين مصفوف بها العدد ٤ أي `int arr[4]` فهذا يعني أننا قمنا بأخذ أربعة أماكن في الذاكر كل واحدة مهنا بحجم المتغير `int`، توجد طريقتين لإعطاء قيمة للمصفوفات، الطريقة الأولى هي كتابة المصفوفة ثم بعدها مباشـــرة القيم التي بها، مثال توضيحي:

```
1  #include<stdio.h>
2
3  main()
4  {
5      int arr[4] = {10, 12, 13, 15};
6
7      printf("[%d][%d][%d][%d]\n", arr[0], arr[1], arr[2], arr[3]);
8  }
```

البرنامج ٢,٣,٣: طريقة إعطاء قيم لمصفوفة

حيث ستكون طريقة وضع القيم منظمة، مثلا إن وضعنا ثلاثة قيم لمصفوفة ذات أربعة أماكن فإن المكان الرابع سيبقى بدون قيمة، و ترى أنه عندما طبعنا النتائج في السطر السابع بدأنا بالمصفوف ٠ حيث جميع المصفوفات تبدأ من ٠، لأن ٠ يعتبر قيمة. لا يمكن استعمال المثال السابق بهذه الطريقة:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int arr[4];
6        arr[4] = {10, 12, 13, 15};
7
8        printf("[%d][%d][%d][%d]\n", arr[0], arr[1], arr[2], arr[3]);
9  }
```

البرنامج ٢,٣,٤: طريقة إعطاء قيم لمصفوفة (٢)

أي أنه يجب إعطاء القيم مباشرة بعد الإعلان عن المصفوفة، توجد طريقة أخرى و هي الإعلان عن المصفوفة ثم إعطاء كل صف قيمته و هذا مثال عن طريقة استعمالها:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int arr[4];
6        arr[0] = 10;
7        arr[1] = 12;
8        arr[2] = 13;
9        arr[3] = 15;
10
11       printf("[%d][%d][%d][%d]\n", arr[0], arr[1], arr[2], arr[3]);
12 }
```

البرنامج ٢,٣,٥: طريقة إعطاء قيم لمصفوفة (٣)

و هي الطريقة التي تستعمل بكثرة.

٢,٣,٢ المصفوفات الثنائية الأبعاد:

المصفوفات التي درسنها سابقا هي مصفوفات ذات بعد واحد، الآن سندرس مصفوفات ذات بوعدين حيث ستكون طريقة الاستعمال بسيطة و مشابه للمصفوفات ذات بوعد واحد. لكتابة مصفوفة ذات بوعدين، سنكتب مصفوفة طبيعية ثم نظيف إليها صف آخر و هذا مثال سأقوم بشرحه:

```
1  #include<stdio.h>
2
3  main()
```

```
4  {
5        int arr2d[2][2];
6        int i,j;
7
8        arr2d[0][0] = 10;
9        arr2d[0][1] = 20;
10
11       arr2d[1][0] = 30;
12       arr2d[1][1] = 40;
13
14       for(i=0;i<=1;i++)
15       {
16             for(j=0;j<=1;j++)
17             {
18                   printf("arr2d[%d][%d] = %d\n", i, j, arr2d[i][j]);
19             }
20       }
21
22 }
```

البرنامج ٢,٣,٦: طريقة الإعلان عن مصفوفات ثنائية الأبعاد

في السطر الخامس قمنا بإنشاء المصفوفة الثنائية، أعطين قيم لكل صف من صفوف المصفوفات في كل من السطر الثامن و التاسع للمصفوفة الأولى [0]، و السطر الحادي عشر و الثاني عشر للمصفوفة الثانية [1]. مصفوفات ثنائية الأبعاد هي المصفوفات بها مصفوفات، أي مصفوفة رئيسية و مصفوفة ثانوية، فإذا كانت لدينا مصفوفة رئيسية بحجم ٢ من نوع أعداد صحيحة و مصفوفة ثانوية بحجم ٢ فهذا يعني أن المصفوفة الرئيسية الأولى تنقصم إلى مصفوفتين ثانويتين، و المصفوفة الرئيسية الثانية أيضا تنقسم إلى مصفوفتين ثانويتين. في المصفوفات الثنائية الأبعاد توجد طريقتين أيضا لإعطائها قيم سابقة، الطريقة الأولى وضعنها في المثال السابق، أما الطريقة الثانية فهي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int arr2d[2][2] = {{10, 20}, {30, 40}};
6        int i,j;
7
8        for(i=0;i<=1;i++)
9        {
10             for(j=0;j<=1;j++)
11             {
12                   printf("arr2d[%d][%d] = %d\n", i, j, arr2d[i][j]);
13             }
14       }
15
16 }
```

البرنامج ٢,٣,٧: طريقة الإعلان عن مصفوفات ثنائية الأبعاد (٢)

و هي مشابها لطريقة استعمالها في المصفوفات ذات بوعد واحد.

٢,٣,٢ المصفوفات الثلاثية الأبعاد:

الآن يمكن حتى عمل مصفوفات ذات أربعة أو خمسة أبعاد، بنفس الطرق السابقة، فمثلا إذا أردنا وضـــع مـــصفوفة ثلاثية الأبعاد و إعطاء لها قيم سابقة أو ترك المستخدم يضع لها قيم فسنكتب الآتي:

في حالة إعطاء قيم سابقة للمصفوفات نكتب:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int arr3d[2][2][2] = {{{10, 20},{30, 40}}, {{50, 60},{70, 80}}};
6        int i, j, k;
7
8        for(i=0;i<=1;i++)
9        {
10             for(j=0;j<=1;j++)
11             {
12                   for(k=0;k<=1;k++)
13                   {
14                         printf("arr3d[%d][%d][%d] = %d\n", i, j, k,
   arr3d[i][j][k]);
15                   }
16             }
17       }
18
19 }
```

البرنامج ٢,٣,٨: طريقة الإعلان عن مصفوفات ثلاثية الأبعاد

و في حالة ترك المستخدم يقوم بإدخال القيم نكتب:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int arr3d[2][2][2];
6        int i, j, k;
7
8        for(i=0;i<=1;i++)
9        {
10             for(j=0;j<=1;j++)
11             {
12                   for(k=0;k<=1;k++)
13                   {
14                         printf("arr3d[%d][%d][%d] : ", i, j, k);
15                         scanf("%d", &arr3d[i][j][k]);
16                   }
17             }
18       }
19
20       for(i=0;i<=1;i++)
21       {
```

```
22              for(j=0;j<=1;j++)
23              {
24                    for(k=0;k<=1;k++)
25                    {
26                          printf("arr3d[%d][%d][%d] = %d\n", i, j, k,
arr3d[i][j][k]);
27                    }
28              }
29        }
30
31  }
```

البرنامج ٢,٣,٩: طريقة الإعلان عن مصفوفات ثلاثية الأبعاد (٢)

٢,٣,٣ مصفوفة ذات حجم غير معروف:

يمكننا إنشاء مصفوفة ذات حجم غير معروف، حيث ستكون المصفوفة ديناميكية الحجم أي أن حجم المصفوفة سيزيد حسب الطلب، و لكن من شروط المصفوفات الديناميكية يجب أن تكون القيم معطاة سابقا أي لا يمكن للمستخدم إدخال قيم في مصفوفة مجهولة الحجم. سأعطي أمثلة حول المصفوفات ذات حجم غير معروف، ثم نناقشها:

```
1   #include<stdio.h>
2
3   main()
4   {
5         int arr[] = {10, 20, 30};
6
7         printf("%d\n", arr[0]);
8         printf("%d\n", arr[1]);
9         printf("%d\n", arr[2]);
10  }
```

البرنامج ٢,٣,١٠: طريقة الإعلان عن مصفوفة ذات حجم غير معروف

في السطر الخامس:

```
5 | int arr[] = {10, 20, 30};
```

هي الطريقة الوحيدة التي يمكن استعمالها حاليا، و لا يمكن استعمالها في مصفوفات ثانوية، أي مثلا في المصفوفات الثلاثية الأبعاد لا يمكن استعمال مثل الآتي:

```
int arr3d[][][] = {{{10, 20},{30, 40}}, {{50, 60},{70, 80}}};
```

الإمكانيات الوحيدة في المصفوفات الثنائية و الثلاثية الأبعاد أو أكثر هي وضع المصفوفة الرئيسية بدون حجم، و كي يصبح المثال السابق صحيح نكتب:

```
int arr3d[][2][2] = {{{10, 20},{30, 40}}, {{50, 60},{70, 80}}};
```

٢,٣,٤ السلاسل الحرفية (النصوص):

الآن سنعرف طريقة التعامل مع النصوص، باستخدام المصفوفات، و تسمى بسلاسل من حروف. و تسمى بسلاسل حرفية لأنها في الحقيقة هي عبارة عن سلاسل بها أماكن و كل مكان به حرف، رمز، أو رقم، توجد طرق كثيرة لتعامل مع السلاسل الحرفية باستخدام المصفوفات منها:

إعطاء حجم لمصفوفة من نوع char حيث يكون حجمها هو الحد الأقصى لعدد الحروف التي يمكن إدخالها، مثال توضيحي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        char text[14] = "Hello, World!";
6
7        printf("%s\n", text);
8  }
```

البرنامج ٢,٣,١١: طريقة الإعلان عن سلسلة حرفية

أو يمكن تجزئة النص كما يلي:

```
1   #include<stdio.h>
2
3   main()
4   {
5         char text[14];
6
7         text[0] = 'H';
8         text[1] = 'e';
9         text[2] = 'l';
10        text[3] = 'l';
11        text[4] = 'o';
12        text[5] = ',';
13        text[6] = ' ';
14        text[7] = 'w';
15        text[8] = 'o';
16        text[9] = 'r';
17        text[10] = 'l';
18        text[11] = 'd';
19        text[12] = '!';
20        text[13] = '\0';
21
22        printf("%s\n", text);
23  }
```

البرنامج ٢,٣,١٢: طريقة الإعلان عن سلسلة حرفية (٢)

في السطر العشرين قمنا بإضافة الرمز \0 و الذي يعني نهاية السلسلة، و عدم وجوده سيعطي نتائج غير مرغوب فيها، أما في الطريقة الأولى فيكفي أن نعطيه مكان إضافي و سيتم إضافته تلقائيا، و إذا انتبهت إلى المثال السابق في:

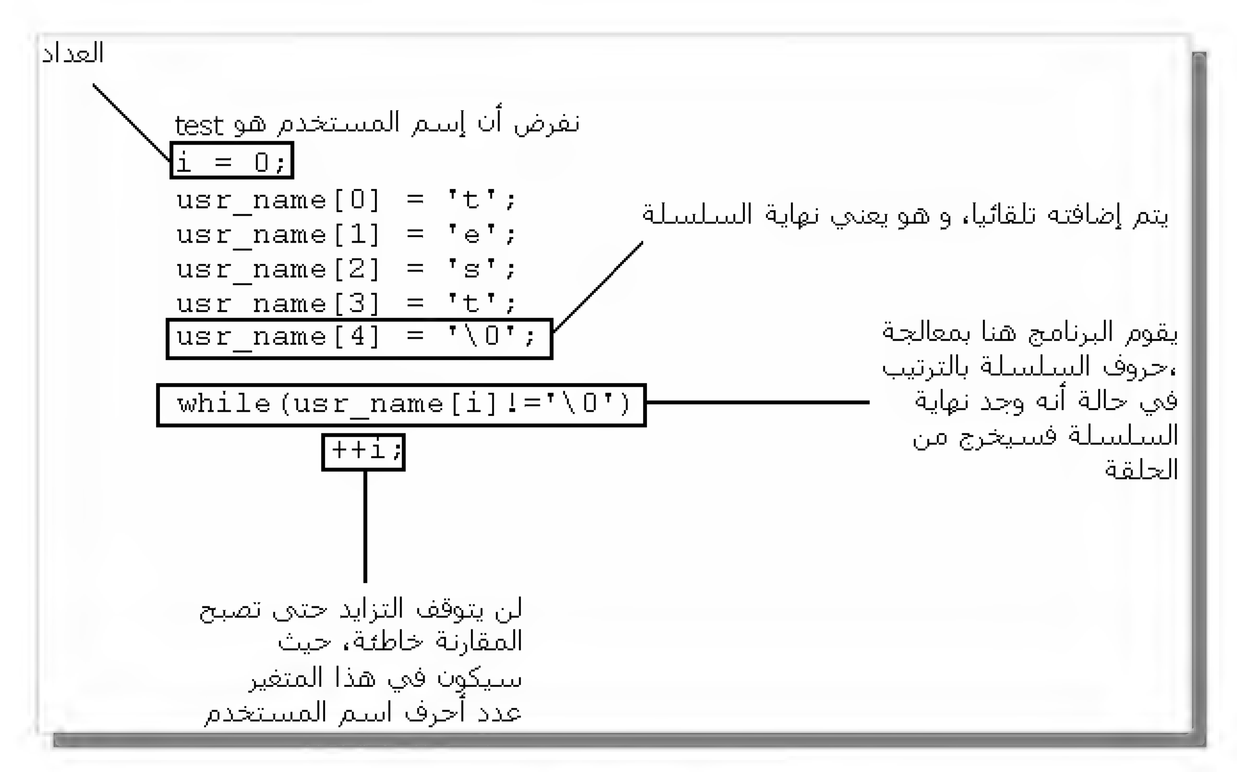
```
5 | char text[14] = "Hello, World!";
```

فستلاحظ أن عدد الأحرف هو ١٣ (الفراغ يعتبر حرف) و نحن قمنا بحجز ١٤ مكان، المكان الرابع عشر سيكون لرمز \0 حيث سيتم إضافته تلقائيا. الآن سنقوم بكتابة برنامج يطلب من المستخدم إدخال اسمه، ثم يطبع له البرنامج عدد أحرف اسمه:

```
1  #include<stdio.h>
2
3  main()
4  {
5      char usr_name[255];
6      int i;
7      i = 0;
8
9      printf("Your name: ");
10     scanf("%s", &usr_name);
11
12     while(usr_name[i] != '\0')
13     {
14         ++i;
15     }
16
17     printf("%s = %d characters\n", usr_name, i);
18 }
```

البرنامج ٢,٣,١٣: حساب عدد أحرف إسم مستخدم

في السطر الخامس قمن بالإعلان عن سلسلة حرفية باسم usr_name، بحجم ٢٥٥ مكان و هو أقصى احتمال يمكن الوصول إليه، حيث ستكون تلك السلسلة هي المكان الذي سنضع فيه اسم المستخدم، و في السطر السادس قمن بالإعلان عن متغير و الذي سيكون العداد لاسم المستخدم، ثم أعطيناه القيمة صفر في السطر السابع، ثم طلبنا من المستخدم إدخال اسمه في السطر التاسع، و خذنا اسم المستخدم في السطر العاشر حيث ستجد الرمز %s و الحرف s مختصر من *string* و هو الرمز الخاص بالسلاسل الحرفية، و في السطر الثاني عشر قمنا بإنشاء حلقة لا تنتهي إلا بانتهاء اسم المستخدم، و كي تفهم هذه الأخيرة إليك الصورة التالية:



الشكل ٢,٣,٢: طريقة وضع البيانات في المصفوفات

و في السطر السابع عشر يقوم البرنامج بطباعة اسم المستخدم و عدد أحرفه. و إذا أردت جعل البرنامج أكثر مرونة و أكثر تعمق جرب هذا المثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5      char usr_name[255];
6      int i;
7      i = 0;
8
9      printf("Your Fullname: ");
10     scanf("%s", &usr_name);
11
12     while(usr_name[i]!='\0')
13     {
14         printf("%i: %c\n", i+1, usr_name[i]);
15         ++i;
16     }
17
18     printf("%s = %d characters\n", usr_name, i);
19 }
```

البرنامج ٢,٣,١٤: حساب عدد أحرف إسم مستخدم (٢)

٢,٣,٤,١ الدالة gets:

تحدثنا عن الدالة gets في الفصل الأول، و لكننا لم نتحدث عن طريقة استعمالها. هي خاص بإدخال النصوص حيث بدل استعمال scanf("%s", &string_name) نستعمل gets(string_name) أفضل، مثال:

```
1  #include<stdio.h>
2  
3  main()
4  {
5      char usr_name[255];
6  
7      puts("Your name:");
8      gets(usr_name);
9  
10     puts("nice to meet you");
11     puts(usr_name);
12 }
```

البرنامج ٢,٣,١٥: الدالة gets

و الفرق بين الدالة scanf و الدالة gets هو عند إدخال الأسماء، في الدالة gets إن كتبت اسمين و فصلت بينهما بفراغ فسيقوم بطباعة كلاهما، أما في الدالة scanf فإنه ستتوقف عند الفراغ الأول و تقوم بطبع ما هو قبل الفراغ لا أكثر.

٢,٣,٤,٢ الدالة strcpy و الدالة strncpy:

الدالة strcpy من دوال الملف الرأسي string.h حيث به مجموعة من الدوال الخاصة بالتعامل مع السلاسل الحرفية، و اسم الدالة مختصر من *string copy*، و هي تقوم بنسخ و لصق الحروف من سلسلة إلى أخرى، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3  
4  main()
5  {
6      char String[] = "String!";
7      char Empty_String[20];
8  
9      strcpy(Empty_String, String);
10 
11     printf("Empty_String = %s\n", Empty_String);
12 }
```

البرنامج ٢,٣,١٦: الدالة strcpy

الوسيط الأول من الدالة نقوم بالكتابة فيه اسم السلسلة الحرفية التي نريد أن ننسخ بها النص، و في الوسيط الثاني نكتب السلسلة التي نريد نسخها. أيضا الدالة strncpy من دوال الملف الرأسي string.h، و هي مثـــل الدالـــة الـــسابقة strcpy مع إضافة بسيطة و هي تحديد عدد الأحرف التي نريد نسخها، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main()
5  {
6        char String[] = "String!";
7        char Empty_String[20];
8
9        strncpy(Empty_String, String, 3);
10
11       printf("Empty_String = %s\n", Empty_String);
12 }
```

البرنامج ٢,٣,١٧: الدالة strncpy

في الوسيط الثالث من الدالة strncpy نقوم بتحديد عدد الأحرف التي نريد نسخها.

و يمكن أيضا كتابة التالي:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main()
5  {
6        char Empty_String[20];
7
8        strcpy(Empty_String, "String!");
9
10       printf("Empty_String = %s\n", Empty_String);
11 }
```

البرنامج ٢,٣,١٨: الدالة strcpy (٢)

٢,٣,٤,٣ الدالة strcat و الدالة strncat:

الدالة strcat من دوال الملف الرأسي string.h، و هي مختصرة من *concatenate string*، و هي تقوم نسخ نص و إضافته في نهاية سلسلة حرفية، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main()
5  {
6        char String[20] = "String!";
7
8        strcat(String, ", String2");
```

```
9
10        printf("String = %s\n", String);
11  }
```

البرنامج ٢,٣,١٩: الدالة strcat

في الوسيط الأول من الدالة strcat نقوم بكتابة اسم السلسلة التي سنضيف إليها النص، و في الوسيط الثاني نقوم بكتابة النص. الدالة strncat مثل الدالة strcat مع إضافة بسيطة و هي تحديد عدد الأحرف التي نريد نسخها، مثال:

```
1   #include<stdio.h>
2   #include<string.h>
3
4   main()
5   {
6         char String[20] = "String!";
7
8         strncat(String, ", String2", 3);
9
10        printf("String = %s\n", String);
11  }
```

البرنامج ٢,٣,٢٠: الدالة strncat

في الوسيط الثالث نقوم بتحديد عدد الأحرف التي نريد نسخها.

٢,٣,٥ طرق أخرى لتعامل مع المصفوفات:

يمكننا كتابة `*(arr+0) = 10` في مكان `arr[0] = 10`، حيث أنها مكافئة للسابقة، و هذا مثال طبيعي:

```
1   #include<stdio.h>
2
3   main()
4   {
5         int arr[1];
6         arr[0] = 10;
7
8         printf("%d\n", arr[0]);
9   }
```

البرنامج ٢,٣,٢١: طرق أخرى لتعامل مع المصفوفات

و هذا المثال السابق باستعمال الطريقة الثانية:

```
1   #include<stdio.h>
2
3   main()
4   {
5         int arr[1];
```

```
6        *(arr+0) = 10;
7
8        printf("%d\n", *(arr+0));
9    }
```

البرنامج ٢,٣,٢٢: طرق أخرى لتعامل مع المصفوفات (٢)

الشفرة arr[0] هي نفسها الشفرة ‎*(arr+0)‎. و يمكن استعمال مؤثرات مثل الجمع، الطرح، القسمة و الضرب للإشارة إلى عنصر من مصفوفة مثل:

```
1   #include<stdio.h>
2
3   main()
4   {
5       int arr[2];
6
7       arr[0+0] = 10;
8       printf("%d\n", arr[0+0]); /* or printf("%d\n", arr[0]); */
9
10      arr[0-0] = 20;
11      printf("%d\n", arr[0-0]); /* or printf("%d\n", arr[0]); */
12
13      arr[1*1] = 30;
14      printf("%d\n", arr[1*1]); /* or printf("%d\n", arr[1]); */
15
16      arr[1/1] = 40;
17      printf("%d\n", arr[1/1]); /* or printf("%d\n", arr[1]); */
18  }
```

البرنامج ٢,٣,٢٣: طرق أخرى لتعامل مع المصفوفات (٣)

٢,٣,٦ الأخطاء المحتملة:

١. نفرض أنه لديك مصفوفة بحجم ٢ و اسمها arr، ثم أردت وضع قيم للمصفوفة و قمت بكتابة التالي:

```
int arr[2] ;
arr[0] = 10 ;
arr[8] = 20 ;
```

فهنا سيحدث انتهاك في الذاكرة، و ربما يؤدي إلى توقف البرنامج عن العمل. و الطرقة الصحيحة للمثال السابق هي:

```
int arr[2] ;
arr[0] = 10 ;
arr[1] = 20 ;
```

٢. في السلاسل الحرفية عند كتابة نص مباشرة بعد الإعلان عنها فيجب عليك دائما وضع مكان إضافي لرمز \0 و إلا ستكون النتائج غير صحيحة.

٣. الدالة gets لا تستعمل إلا في إدخال النصوص.

٢,٣,٧ تمارين:

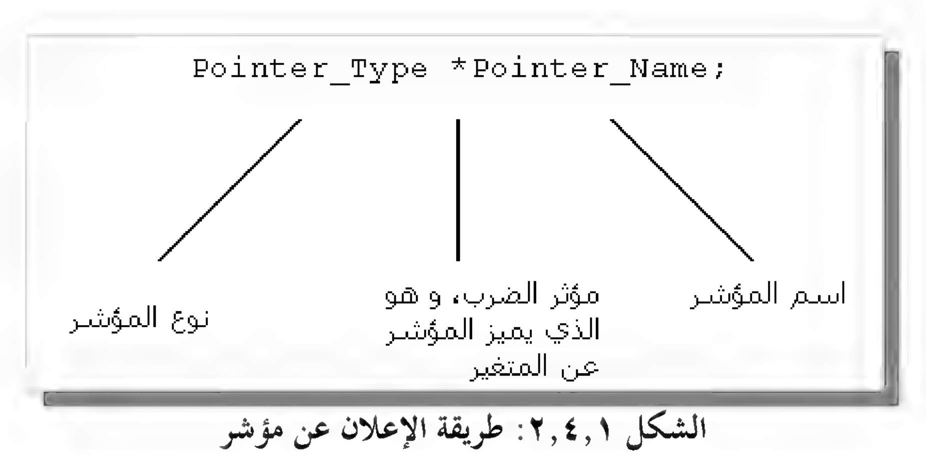
١. أكتب برنامج يطلب من المستخدم إدخال اسمه، ثم يطبع له البرنامج عدد أحرف اسمه (باستخدام الدالتين puts و gets).

٢. أكتب برنامج يطلب من المستخدم إدخال أعداد حقيقية، ثم يقوم البرنامج بإعادة طبعها (باستخدام المصفوفات).

٣. أكتب برنامج به مصفوفة بها النص *Hello, World !* ثم قم بعمل حلقة تقوم بعد عدد رموز أو أحرف تلك الجملة(إذا وجدت ١٣ حرفا فإن النتيجة خاطئة) يجب أن تكون النتيجة ١٢ مع طريقة استعمال سليمة يعني بدون استعمال -١.

٢,٤ المؤشرات Pointers

كلمة مؤشر تعني الإشارة إلى شيء، و في لغة C المؤشرات تشير إلى عناوين في الذاكرة. طريقة الإعلان عن مؤشر مماثلة لطريقة الإعلان عن المتغيرات، و الإختلاف بين المؤشر و المتغير هو أن المؤشر يشير إلى عنوان متغير أو عنوان عشوائي في الذاكرة. و توجد ملاحظة هنا و هي أن المؤشرات في الأنظمة الحديث لا يمكن إعطائها عناوين عشوائية (إلا في حالات خاصة) و ذلك لتشدد الحماية فيها. لذا هنا سنرى فقط طريقة التعامل مع المؤشرات و المتغيرات، و هذه صورة توضح لطريقة الإعلان عن مؤشر:



الشكل ٢,٤,١: طريقة الإعلان عن مؤشر

الفرق الوحيد بين الإعلان عن متغير و الإعلان عن مؤشر هو مؤثر الضرب في المؤشرات و الذي يكون قبل اسم المؤشر.

٢,٤,١ نوع المؤشر Pointer Type:

نوع المؤشر يكون حسب رغبنا، مثلا لو أردنا الإشارة إلى متغير حجمه ٢ بايت فيجب علينا الإعلان عن مؤشر يمكنه الوصول إلى ٢ بايت من الذاكرة. للمؤشرات أنواع هي نفسها أنواع المتغيرات،و هي int, float, double, long, short, unsigned, signed, char,.

٢,٤,٢ اسم المؤشر Pointer Name:

لاسم المؤشر شروط هي نفسها شروط المتغيرات و هي:

- أن لا يتجاوز اسم المؤشر أكثر من ٣١ حرف.
- أن لا يبدأ اسم المؤشر بأرقام.
- أن لا يكون اسم المؤشر يحتوي على مؤثرات مثل الجمع و الطرح و....
- أن لا يكون اسم المؤشر يحتوي على رموز مثل % و # و } و...(باستثناء الرمز _).

- أن لا يكون اسم المؤشر مستعمل سابقا في دالة أو متغير أو مؤشر آخر.
- أن لا يكون اسم المؤشر من أسماء الكلمات المحجوزة.

المؤشرات تحمل عناوين لمواقع في الذاكرة و لا يمكن أن نعطيها قيم مباشرة إلا في حالات، و في العناوين نجد قيم و لكل عنوان قيمته، مثال:

```
1  #include<stdio.h>
2  
3  main()
4  {
5  	int *ptr, i;
6  	i = 10;
7  	ptr = &i;
8  
9  	printf("*ptr = %d\n", *ptr);
10 }
```

البرنامج ٢,٤,١: طريقة الإعلان عن مؤشر

المؤشر موجود في السطر الخامس مع متغير، اسم المؤشر هو ptr و اسم المتغير هو i، أعطينا للمتغير i القيمة ١٠ في السطر السادس، و في السطر السابع أعطينا للمؤشر ptr عنوان المتغير i أي أنه عند كتابة &Variable_Name فهذا يعني عنوان متغير، و أخيرا السطر التاسع حيث كتبنا *ptr و الذي يعني القيمة الموجود في عنوان المؤشر ptr، و في حالة أننا كتبنا اسم المؤشر بدون مؤثر الضرب فسيتم طباعة عنوان المؤشر ptr. في المثال السابق إن قمنا بتغير القيمة الموجودة في عنوان المؤشر ptr فستتغير القيمة الموجود في المتغير i، لأننا أخذنا عنوان المتغير i و الذي هو نفسه عنوان المؤشر ptr، جرب المثال السابق بهذه الطريقة:

```
1  #include<stdio.h>
2  
3  main()
4  {
5  	int *ptr, i;
6  	i = 10;
7  	ptr = &i;
8  	*ptr = 100;
9  
10 	printf("*ptr = %d\n", *ptr);
11 }
```

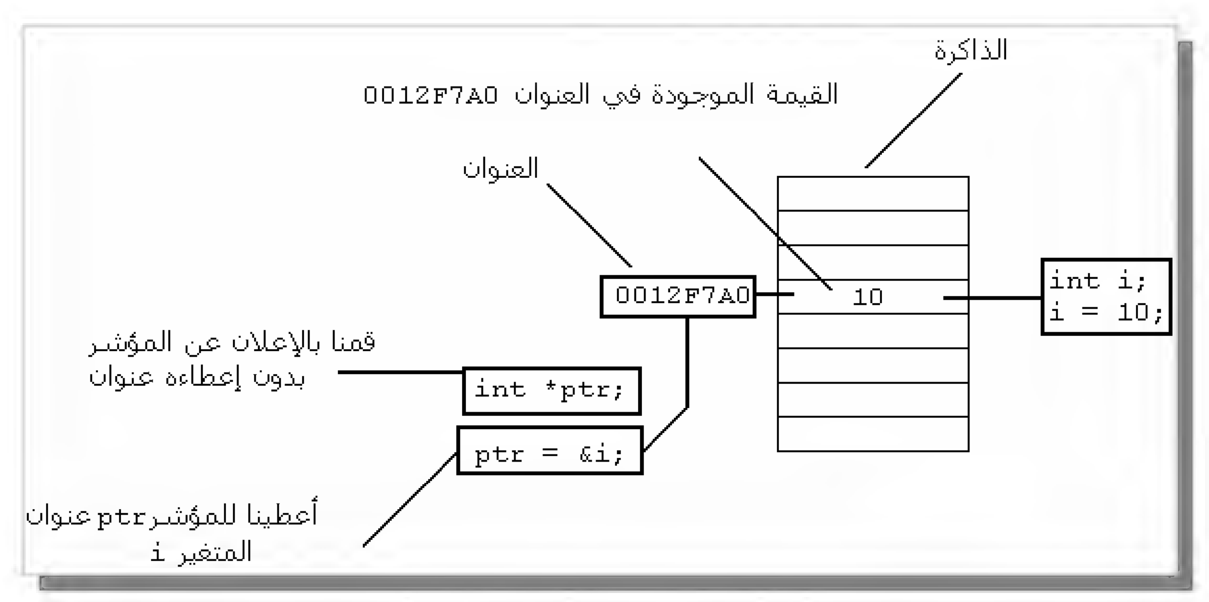
البرنامج ٢,٤,٢: طريقة إستعمال مؤشر

هنا يمكننا إعطاء قيمة للمؤشر ptr لأنه لديه عنوان و هو عنوان المتغير i، و سيقوم بحذف القيمة السابقة الموجود في عنوان i و يقوم بتحديثها إلى العدد ١٠٠. و لكي تفهم المؤشرات جيدا جرب المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        int *ptr, i;
6        i = 10;
7        ptr = &i;
8
9        printf("i = %d\n", i);
10       printf("*ptr = %d\n", *ptr);
11       printf("&i = %p\n", &i);
12       printf("ptr = %p\n", ptr);
13 }
```

البرنامج ٢,٤,٣: طريقة إستعمال مؤشر (٢)

في السطر التاسع و العاشر قمنا بطباعة كل من القيمة الموجودة في المتغير i و القيمة الموجود في عنوان المؤشر ptr و لكي نرى قيمة موجودة داخل مؤشر نكتب مؤثر الضرب قبل اسمه و أيضا يمكننا وضع قيمة له بنفس الطريقة (يجب أن يكون لديه عنوان كي نستطيع وضع له قيمة)، و في السطر الحادي عشر و السطر الثاني عشر قمنا بطباعة كل من عنوان المتغير i و عنوان المؤشر ptr، و تلاحظ أنه وضعنا الرمز %p و هو مختصر من *pointer* و يمكن استعماله مع المؤشرات أو عنوان لمتغير حيث قمنا باستعماله كي يتم طباعة العناوين بشكل صحيح. صورة توضيحية للمثال السابق:



الشكل ٢,٤,٢: الذاكرة و العناوين

٢,٤,٣ المؤشرات و المصفوفات:

المؤشرات شبيها بالمصفوفات، لأننا نشير إلى عنوان يمكننا التقدم به (بإستخدام المؤثر ++ مثلا) مما يجعلنا نصل إلى مجموعة من بيتات متسلسلة. و هنا مثال يوضح الشبه بين المؤشرات و المصفوفات:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int arr[2];
6          int *ptr;
7
8          arr[0] = 10;
9          arr[1] = 20;
10
11         ptr = &arr[0];
12
13         printf("%d\n", ptr[0]);
14         printf("%d\n", ptr[1]);
15 }
```

البرنامج ٢,٤,٤: إستعمال المؤشر على طريقة المصفوفات

قمنا بالإعلان عن مصفوفة بحجم ٢، ثم قمنا بالإعلان عن مؤشر، و في السطر الثامن و التاسع أعطينا قيم للمصفوفة، و في السطر الحادي عشر أعطينا للمؤشر ptr عنوان بداية المصفوفة، و في السطر الثالث عشر و الرابع عشر قمنا بطباعة ما هو موجود في عنوان المؤشر ptr. و يمكننا كتابة المثال السباق بهذه الطريقة:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int arr[2];
6          int *ptr;
7
8          *(arr+0) = 10;
9          *(arr+1) = 20;
10
11         ptr = &*(arr+0);
12
13         printf("%d\n", *(ptr+0));
14         printf("%d\n", *(ptr+1));
15 }
```

البرنامج ٢,٤,٥: طريقة أخرى لتعامل مع المؤشرات

و يمكن أيضا كتابته بالطريقة التالية:

```
1  #include<stdio.h>
2
3  main()
4  {
5          int arr[2];
6          int *ptr;
7
8          arr[0] = 10;
9          arr[1] = 20;
```

```
10
11        ptr = &arr[0];
12
13        printf("%d\n", *ptr);
14        printf("%d\n", *++ptr);
15 }
```

البرنامج ٢,٤,٦: طريقة أخرى لتعامل مع المؤشرات (٢)

و هذا مثال لا يمكن تطبيقه على المصفوفات:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int arr[2];
6         int *ptr;
7
8         arr[0] = 10;
9         arr[1] = 20;
10
11        ptr = &arr[0];
12
13        printf("%d\n", arr);
14        printf("%d\n", ++arr);
15 }
```

البرنامج ٢,٤,٧: إمكانيات المؤشر مقارنة مع المصوفات

٢,٤,٤ التعامل مع النصوص:

التعامل مع النصوص باستخدام المؤشرات مشابه للتعامل مع النصوص باستخدام المصفوفات،مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5         char *str = "Hello, World!";
6
7         printf("%s\n", str);
8  }
```

البرنامج ٢,٤,٨: التعامل مع النصوص بإستخدام المؤشرات

أما إذا أردنا استخدام المؤشرات في الإدخال فيوجد شروط يجب التقيد بها و إلا ستحدث أخطاء ربما في المصدر البرنامج أو أثناء تشغيل البرنامج، فمثلا لا يمكننا استعمال مثل ما هو في المثال التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5         char *str;
```

```
6
7        printf("Enter a string: ");
8        scanf("%s", str);
9        printf("%s\n", str);
10 }
```

البرنامج ٢,٤,٩: التعامل مع النصوص بإستخدام المؤشرات (٢)

في المترجمات الجديد ربما لن يسمح لك المترجم باستعمال هذه الطريقة، و السبب في ذلك هو عدم تحديد للمؤشر str عنوانا، أي أنه بلا عنوان، و لكي تصبح طريتنا صحيح فيجب على الأقل الإعلان عن متغير حرفي و نعطي للمؤشر str عنوان ذلك المتغير و الذي سيكون بداية المؤشر str و سيصبح المثال على الشكل التالي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        char *str;
6        char adr_str;
7
8        str = &adr_str;
9
10       printf("Enter a string: ");
11       scanf("%s", str);
12
13       printf("%s\n", str);
14 }
```

البرنامج ٢,٤,١٠: التعامل مع النصوص بإستخدام المؤشرات (٣)

و ستلاحظ في السطر الحادي عشر قمنا بكتابة str بدون الرمز &، ذلك لأننا استعملنا مؤشر و الوضع الافتراضي للمؤشرات هي عنوالها. و يمكننا أيضا كتابة اسم مصفوفة بدون رمز العنوان & في الدالة scanf، مثال توضيحي:

```
1  #include<stdio.h>
2
3  main()
4  {
5        char arr[255];
6
7        printf("Enter a string: ");
8        scanf("%s", arr);
9
10       printf("%s\n", arr);
11 }
```

البرنامج ٢,٤,١١: التعامل مع النصوص بإستخدام المؤشرات (٤)

و من إمكانيات المؤشرات أنه يمكن أن نعطيه قيم مصفوفة باستخدام:

```
1  #include<stdio.h>
2
```

```
3  main()
4  {
5        int arr[3] = {10, 20, 30};
6        int *ptr;
7        int i;
8
9        ptr = arr;
10
11       for(i=0;i<=2;i++)
12             printf("ptr[%d] = %d\n", i, ptr[i]);
13 }
```

البرنامج ٢,٤,١٢: التعامل مع النصوص بإستخدام المؤشرات (٥)

و لا يمكننا كتابة العكس، `arr = ptr`.

٢,٤,٥ المرجع reference:

المرجع هو أخذ عنوان متغير، عند كتابة:

```
1  #include<stdio.h>
2
3  main(){
4        int ref;
5        int *ptr;
6        ref = 10;
7        ptr = &ref;
8  }
```

البرنامج ٢,٤,١٣: المرجع

هنا الـــ `ptr` هو المؤشر، و الـــ `&ref` هو المرجع. كتابة المؤثر `&` ثم اسم متغير يعني أخذ عنوان ذلك المتغير، و تـــسمى هذه العملية بالمرجع *reference*.

٢,٤,٦ مؤشر لـــ void:

لا يمكننا الإعلان عن متغير باستخدام الكلمة المحجوزة `void`، وسبب في ذلك أنه ليس لها حجم كي يتم الحفظ فيه القيم المرادة، و لكننا يمكن أن نقوم بالإعلان عن مؤشر لـــ `void` حيث يعتبر الأكثر مرونة مع المؤشرات، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        void *p_void;
6        /* can't use void v_void; */
7
8        (int)p_void = 10;
9        printf("(int)p_void = %d\n", p_void);
10
```

```
11        (char)p_void = 'H';
12        printf("(char)p_void = %c\n", p_void);
13
14        (char *)p_void = "Hello";
15        printf("(char *)p_void = %s\n", p_void);
16    }
```

البرنامج ٢,٤,١٤: مؤشر لـ void

عندما نريد وضع قيم لمؤشر void نكتب `(Type)Pointer_Name`، في مكان Type نكتب نوع القيمة التي سنقوم بإدخالها، ثم اسم المؤشر ثم نعطيه القيمة.

٢,٤,٧ مؤشر لمصفوفة:

عندما نعلن عن مؤشر لمصوفة فهذا يعني أن كل عنصر من تلك المصفوفة يمكن أي يحمل عنوانا، مثال:

```
1    #include<stdio.h>
2
3    main()
4    {
5        int *arr[2];
6        int a = 10, b = 20;
7
8        printf("A = %d, ", a);
9        printf("B = %d\n", b);
10
11       arr[0] = &a;
12       *arr[0] = 5;
13
14       arr[1] = &b;
15       *arr[1] = 10;
16
17       printf("A = %d, ", a);
18       printf("B = %d\n", b);
19   }
```

البرنامج ٢,٤,١٥: مؤشر لمصفوفة

و يمكن استعمال مؤشر لمصفوفة ثنائية أو ثلاثية الأبعاد. و يمكن أيضا استعمال مؤشر لسلسلة حرفية حيث كل عنصر من تلك السلسلة يمكن أن يحمل نص مثل:

```
1    #include<stdio.h>
2
3    main(){
4        char *arr[] = {"Text 1", "Text 2", "Text 3"};
5
6        printf("arr[0] = %s\n", arr[0]);
7        printf("arr[1] = %s\n", arr[1]);
8        printf("arr[2] = %s\n", arr[2]);
9    }
```

البرنامج ٢,٤,١٦: مؤشر لمصفوفة (٢)

٢,٤,٨ مؤشر لمؤشر:

مؤشر لمؤشر قليلة الاستعمال. مؤشر لمتغير يعني أن نشير إلى عنوان المتغير، و مؤشر لمؤشر يعني أن نشير إلى عنوان مؤشر، مثال:

```
1  #include<stdio.h>
2  
3  main()
4  {
5         int p;
6         int *pt;
7         int **ptr;
8  
9         p = 10;
10        printf("p = %d\n", p);
11 
12        pt = &p;
13        ptr = &pt;
14        **ptr = 5;
15 
16        printf("p = %d\n", p);
17 }
```

البرنامج ٢,٤,١٧: مؤشر لمؤشر

تم الإعلان عن مؤشر لمؤشر في السطر السابع. و يمكن أن نقوم بإعلان عن مؤشر لمؤشر من نوع الحرفي حيث كل مؤشر يمكن أن يحمل سلسلة من حروف مثل:

```
1  #include<stdio.h>
2  
3  main(){
4         char *AdrPtr;
5         char **ptr = &AdrPtr;
6  
7         ptr[0] = "Text 1";
8         ptr[1] = "Text 2";
9         ptr[2] = "Text 3";
10 
11        printf("ptr[0] = %s\n", ptr[0]);
12        printf("ptr[1] = %s\n", ptr[1]);
13        printf("ptr[2] = %s\n", ptr[2]);
14 }
```

البرنامج ٢,٤,١٨: مؤشر لمؤشر (٢)

و يمكن عمل أكثر من مؤشر لمؤشر، أي يمكن الإعلان عن `int *****ptr`، أو أكثر.

٢,٤,٩ الأخطاء المحتملة:

١. في المترجمات الجديد لا يسمح بإعطاء قيمة لمؤشر بدون عنوان، أي لا يمكن كتابة كما في المثال التالي:

```
1 #include<stdio.h>
2
3 main()
4 {
5       int *ptr;
6       *ptr = 10;
7
8       printf("%d\n", *ptr);
9 }
```

البرنامج ٢,٤,١٩: الخطأ ١

و أيضا لا يمكن كتابة:

```
1 #include<stdio.h>
2
3 main()
4 {
5       int *ptr;
6       int i;
7       *ptr = i;
8
9 }
```

البرنامج ٢,٤,٢٠: الخطأ ٢

٢. لا يمكن إعطاء عنوان لمتغير طبيعي:

```
1 #include<stdio.h>
2
3 main()
4 {
5       int *ptr;
6       int i;
7       i = ptr;
8
9 }
```

البرنامج ٢,٤,٢١: الخطأ ٣

٢,٤,١٠ تمارين:

١. أكتب البرنامج التالي باستخدام المؤشرات:

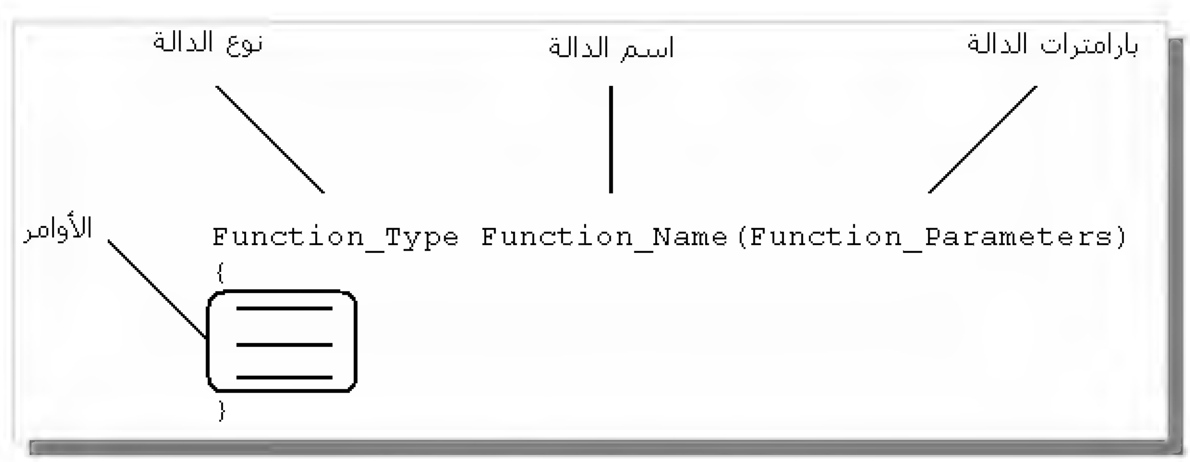
```
1 #include<stdio.h>
2
3 main()
```

```
4  {
5          char usr_name[255];
6          int i;
7          i = 0;
8  
9          printf("Your Fullname: ");
10         scanf("%s", &usr_name);
11 
12         while(usr_name[i]!='\0')
13         {
14               printf("%i: %c\n", i+1, usr_name[i]);
15               ++i;
16         }
17 
18         printf("%s = %d characters\n", usr_name, i);
19 }
```

البرنامج ٢,٤,٢٢: التمرين ١

٢,٥ الدوال Functions

الدوال هي مجموعة من الأوامر و البيانات تحت اسم واحد حيث يمكن استدعاءها من أماكن مختلفة في البرنامج. و هي بما تعرف بـــالروتين الثانوي *subroutine*، و من فوائدها التقليل في شفرة البرنامج و حجم البرنامج، مما يجله أكثرا تنظيما. الصورة التالية توضح طريقة الإعلان عن دالة:



الشكل ٢,٥,١: طريقة الإعلان عن دالة

و يمكن أن لا تحتوي الدالة على وسائط. إذا أردنا عمل دالة تقوم بطباعة الجملـــة *Hello, World !* فـــسيكون برنامجنـــا كالتالي:

```
1   #include<stdio.h>
2
3   void Func_HelloWorld()
4   {
5          printf("Hello, World!\n");
6   }
7
8   main()
9   {
10         Func_HelloWorld();
11  }
```

البرنامج ٢,٥,١: طريقة الإعلان عن دالة

هذه طريقة، أما الطريقة الثانية فهي الإعلان عن الدالة (يسمى بالنموذج *prototype*) ثم نقوم بإعطائها الأوامر بعد الدالة الرئيسية و سيصبح المثال السابق كالتالي:

```
1   #include<stdio.h>
2
3   void Func_HelloWorld();
```

```
4
5   main()
6   {
7         Func_HelloWorld();
8   }
9
10  void Func_HelloWorld()
11  {
12        printf("Hello, World!\n");
13  }
```

البرنامج ٢,٥,٢: طريقة الإعلان عن دالة (٢)

و هي الطريقة الأفضل من حيث التنظيم. و توجد طريقة أخرى و لكن لا يفضل استعمالها من ناحية التنظيف و أيضا بعض المترجمات لا تقبلها و هي:

```
1   #include<stdio.h>
2
3   main()
4   {
5         Func_HelloWorld();
6   }
7
8   void Func_HelloWorld()
9   {
10        printf("Hello, World!\n");
11  }
```

البرنامج ٢,٥,٣: طريقة الإعلان عن دالة (٣)

و إن كان مترجمك قد نبهك عن وجود خطأ فسيكون عن الخطأ عن نموذج *prototype* الدالة `Func_HelloWorld`، و ذلك في الأصل هذه الطريقة هي من طرق لغة *C*، يعني أنها طريقة صحيحة فقط بعض المترجمات لا تدعمها. الكلمة المحجوزة `void` تستعمل مع الدوال حيث حجمها ۰ بايت، و هي لا تقوم بإرجاع أي قيم. سنقوم بكتابة دالة بها وسيط عبارة عن سلسلة حروف ثابتة، حيث ستقوم تلك الدالة بطباعة ما هو داخل الوسيط الخاص بها:

```
1   #include<stdio.h>
2
3   void Func_Print(const char *str);
4
5   main()
6   {
7         Func_Print("Func_Print:\n");
8         Func_Print("Hello, World!\n");
9   }
10
11  void Func_Print(const char *str)
12  {
13        printf("%s", str);
14  }
```

البرنامج ٢,٥,٤: طريقة الإعلان عن دالة ذات وسيط

هذه ليست إلا أمثلة بسيطة حول طريقة عمل الدوال، يمكن أن نقوم بعمل دالة تقوم بـــالجمع، الطـــرح، القـــسمة و الضرب، سيكون المثال على الشكل التالي:

```
1   #include<stdio.h>
2
3   void Func_Add(const int num1, const int num2);
4   void Func_Sub(const int num1, const int num2);
5   void Func_Mul(const int num1, const int num2);
6   void Func_Div(const int num1, const int num2);
7
8   main()
9   {
10          Func_Add(30, 10);
11          Func_Sub(30, 10);
12          Func_Mul(30, 10);
13          Func_Div(30, 10);
14  }
15
16  void Func_Add(const int num1, const int num2)
17  {
18          printf("%d + %d = %d\n", num1, num2, num1+num2);
19  }
20
21  void Func_Sub(const int num1, const int num2)
22  {
23          printf("%d - %d = %d\n", num1, num2, num1-num2);
24  }
25
26
27  void Func_Mul(const int num1, const int num2)
28  {
29          printf("%d * %d = %d\n", num1, num2, num1*num2);
30  }
31
32
33  void Func_Div(const int num1, const int num2)
34  {
35          printf("%d / %d = %d\n", num1, num2, num1/num2);
36  }
```

البرنامج ٢,٥,٥: طريقة الإعلان عن دالة ذات وسيطين

و يمكن أن نجعل هذا البرنامج أكثر مرونة بالطريقة التالية:

```
1   #include<stdio.h>
2
3   void Func_(const int num1, const char sign, const int num2);
4
5   main()
6   {
7           Func_(30, '+', 10);
8           Func_(30, '-', 10);
9           Func_(30, '*', 10);
```

```
10        Func_(30, '/', 10);
11  }
12
13  void Func_(const int num1, const char sign, const int num2)
14  {
15        switch(sign)
16        {
17        case '+':
18              printf("%d %c %d = %d\n", num1, sign, num2, num1+num2);
19              break;
20        case '-':
21              printf("%d %c %d = %d\n", num1, sign, num2, num1-num2);
22              break;
23        case '*':
24              printf("%d %c %d = %d\n", num1, sign, num2, num1*num2);
25              break;
26        case '/':
27              printf("%d %c %d = %d\n", num1, sign, num2, num1/num2);
28              break;
29
30        default:
31              printf("ERROR!\n");
32              break;
33        }
34  }
```

البرنامج ٢,٥,٦: طريقة الإعلان عن دالة ذات أكثر من وسيطين

٢,٥,١ نوع الدالة Function Type:

لــ الدوال أنواع و هي نفسها أنواع المتغير، يمكن استعمال دالة من نوع أعداد صحيحة int أو أعـداد حقيقيـة float. بالنسبة لدوال من نوع أعداد صحيحة فهي لها قيمة تقوم بإرجاعها، أي في نهاية الدالة نقوم بإرجـاع قيمـة باستخدام الكلمة المحجوزة return كما في المثال التالي:

```
1   #include<stdio.h>
2
3   int Int_Func(const int num);
4
5   main()
6   {
7         printf("%d\n", Int_Func(5));
8   }
9
10  int Int_Func(const int num)
11  {
12        return num;
13  }
```

البرنامج ٢,٥,٧: إعلان عن دالة من نوع عدد صحيح

هنا قمنا بإرجاع قيمة الوسيط num إلى الدالة Int_Func، و في السطر السابع، في الدالة printf يمكننا كتابـة دوال التي تقوم بإرجاع قيم كما في هذا المثال، و لا يمكن استعمال هذه الطريقة مع الكلمة المحجوزة void لأنها بدون

حجم و لا يمكنها حمل قيم. يمكن كتابة الدالة بدون نوع بالنسبة للمثال السابقة، لأن دالتنا من نوع أعداد صحيحة، و في لغة C الوضع الافتراضي لدوال بدون نوع هو int و المثال السابق سيصبح على الشكل التالي:

```
1   #include<stdio.h>
2
3   Int_Func(const int num);
4
5   main()
6   {
7         printf("%d\n", Int_Func(5));
8   }
9
10  Int_Func(const int num)
11  {
12        return num;
13  }
```

البرنامج ٢,٥,٨: الوضع الإفتراضي لدالة بدون تحديد نوعها

و يمكن أيضا كتابة اسم متغير بدون نوع، ثم نقوم بكتابة نوعه أسفل اسم الدالة التي هي بعد الدالة الرئيسية، مثال:

```
1   #include<stdio.h>
2
3   Int_Func(const int num);
4
5   main()
6   {
7         printf("%d\n", Int_Func(5));
8   }
9
10  Int_Func(num)
11  const int num;
12  {
13        return num;
14  }
```

البرنامج ٢,٥,٩: طريقة أخرى للإعلان عن وسيط لدالة

و يمكن استعمال المثال:

```
1   #include<stdio.h>
2
3   int Int_Func(const int num);
4
5   main()
6   {
7         printf("%d\n", Int_Func(5));
8   }
9
10  int Int_Func(const int num)
11  {
12        return num;
```

```
13 }
```

البرنامج ٢,٥,١٠: الطريقة الإفتراضية للإعلان عن وسيط لدالة

باستخدام short:

```
1  #include<stdio.h>
2  
3  short Short_Func(const short num);
4  
5  main()
6  {
7        printf("%d\n", Short _Func(5));
8  }
9  
10 short Short _Func(const short num)
11 {
12       return num;
13 }
```

البرنامج ٢,٥,١١: إعلان عن دالة من نوع short

و باستخدام كل من long، float، double، signed، unsigned بنفس الطريقة، أما char فهذا مثال لطريقة استعمالها:

```
1  #include<stdio.h>
2  
3  char Char_Func(const char ch);
4  
5  main()
6  {
7        printf("%c\n", Char_Func('C'));
8  }
9  
10 char Char_Func(const char ch)
11 {
12       return ch;
13 }
```

البرنامج ٢,٥,١٢: إعلان عن دالة من نوع char

و الدوال التي ترجع قيم يمكن إعطاءها لمتغير طبيعي مباشرة مثل:

```
1  #include<stdio.h>
2  
3  int Int_Func(const int num);
4  
5  main()
6  {
7        int i = Int_Func(5);
8  
9        printf("i = %d\n", i);
10 }
```

```
11
12  int Int_Func(const int num)
13  {
14          return num*2;
15  }
```

البرنامج ٢,٥,١٣: إعطاء لمتغير قيمة ترجعها دالة

و يمكن أيضا عمل دالة تقوم بإرجاع سلسلة حرفية باستخدام المؤشرات، مثال:

```
1   #include<stdio.h>
2
3   char *string(char *str){
4           return str;
5   }
6
7   main()
8   {
9           printf("%s\n", string("Hello, World!"));
10  }
```

البرنامج ٢,٥,١٤: إعلان عن دالة من نوع *char

٢,٥,٢ اسم الدالة Function Name:

لاسم الدالة حدود و هي مثل اسم المتغير:

- أن لا يتجاوز اسم الدالة ٣١ حرف.
- أن لا يبدأ اسم الدالة بأرقام.
- أن لا يكون اسم الدالة يحتوي على مؤثرات مثل الجمع و الطرح و....
- أن لا يكون اسم الدالة يحتوي على رموز مثل % و # و { و...(باستثناء الرمز _).
- أن لا يكون اسم الدالة مستعمل سابقا في متغير أو دالة أخرى.
- أن لا يكون اسم الدالة من أحد أسماء الكلمات المحجوزة.

٢,٥,٣ وسائط الدالة Function Parameters:

الوسائط هي متغيرات نقوم بوضعها على حسب متطلباتنا حيث تكون من الأنواع unsigned , signed, short, long, int, float, double, char, char*, char[]. و يمكن أيضا أن تكون الوسائط عبارة عن مصفوفات أو مؤشرات من كل الأنواع.

٢,٥,٤ الأوامر:

الأوامر يمكن كتابتها بحرية مثل ما نكتبها على الدالة الرئيسية main.

٢,٥,٥ المختصرات macros:

لا تسمى بدوال و لكنها شبيها لها، تسمى بالــ *macros* أي المختصرات، و هذا مثال لطريقة الإعلان عنها:

```
1  #include<stdio.h>
2
3  #define Add(a, b) (a+b)
4
5  main()
6  {
7        int a, b;
8        a = 50;
9        b = 100;
10
11       printf("%d + %d = %d\n", a, b, Add(a, b));
12 }
```

البرنامج ٢,٥,١٥:طريقة الإعلان عن مختصر

و هنا عند استعمال متغيرات لا يمكن كتابة أو تحديد نوعها، سيتم تحديد نوع المتغير تلقائيا من الموجه `#define`. مثال آخر:

```
1  #include<stdio.h>
2
3  #define PrintF_(String) printf("%s", String)
4
5  main()
6  {
7        PrintF_("Macro...\n");
8  }
```

البرنامج ٢,٥,١٦: إستدعاء دالة من مختصر

٢,٥,٦ الفرق بين الإجراء و الدالة:

في لغة *C* الإجراءات يمكن القول عليها هي نفسها الدوال لأنها مدمج معها، و الإجراء *Procedure* هو دالة لا تقــوم بإرجاع قيمة و يمكن القول أن دوال من نوع `void` تسمى إجراء لأنها لا ترجع قيم مثل الــدوال مــن نــوع `int` أو `float` أو غيرها، حيث تقوم الإجراءات بتنفيذ أوامر أما الدوال فهي تقوم بعمليات و تعطي نتيجة.

٢,٥,٧ دوال لها وسائط من نوع دوال:

يمكننا عمل دوال بها دوال أخرى، و ذلك كتالي:

```
1  #include<stdio.h>
2
3  void CallFunc(void Func_Name(void));
```

```
4   void Function();
5
6   main(){
7         CallFunc(Function);
8   }
9
10  void CallFunc(void Func_Name(void)){
11        printf("Call Function:\n");
12        Func_Name();
13  }
14
15  void Function(){
16        printf("This is a function!\n");
17  }
```

البرنامج ٢,٥,١٧: دالة ذات وسيط لدالة أخرى

و يمكن أن تكون الوسائط عبارة عن دوال ترجع قيم، أو دوال بها وسائط أخرى، أو استدعاء دوال أخرى في وسائط الدوال، و أكثر من ذلك. و هذا مثال لطريقة استعمال بارامارت دوال لها وسائط أخرى:

```
1   #include<stdio.h>
2
3   void CallFunc(void Func_Name(int a, int b));
4   void Function(int a, int b);
5
6   main(){
7         CallFunc(Function);
8   }
9
10  void CallFunc(void Func_Name(int a, int b)){
11        printf("Call Function:\n");
12        Func_Name(10, 20);
13  }
14
15  void Function(int a, int b){
16        printf("%d + %d = %d\n", a, b, a+b);
17  }
```

البرنامج ٢,٥,١٨: دالة ذات وسيط لدالة أخرى ذات وسائط

٢,٥,٨ الأخطاء المحتملة:

١. لا نضع الفواصل المنقوطة في نهاية دوال نقوم بإعطائها أوامر، مثال:

```
1   #include<stdio.h>
2
3   void Function();
4
5   main()
6   {
7          Function();
8   }
9
10  void Function();
```

```
11 {
12     /*Empty Funtion;*/
13 }
```

البرنامج ٢,٥,١٩: الخطأ ١

و الخطأ في السطر العاشر، عند الإعلان عن دالة بدون أوامر يجب كتابة الفاصلة المنقوطة في نهاية الدالة، أما عند الإعلان عن دوال لكي نعطي لها أوامر فلا يجب أن نكتب فاصلة منقوطة في نهاية الدالة.

٢. لا يمكن الإعلان عن دالتين بنفس الاسم.

٢,٥,٩ تمارين:

١. أكتب دالة تقوم بـــ عد عدد أحرف نص، حيث يكون اسم الدالة StrLen و بها الوســـيط const char *str.

٢. أكتب دالة تقوم بإخراج القيمة المطلقة للعدد الذي أدخله المستخدم، اسم الدالة هو abs مع الوســـيط int value.

٢,٦ الملفات الرأسية Header files

كل من stdio.h و conio.h عبارة عن ملفات رأسية، حيث توجد بها ثوابت، نماذج دوال و بنيات تساعدنا في برامجنا، سنتحدث في هذا الدرس عن طريقة إنشاء ملفات رأسية و طريقة استعمالها. سنقوم بكتابة ملف رأسي بـه دوال لكل من الجمع، الطرح، القيمة و الضرب، أولا نقوم بإنشاء ملف نصي و حفظه على صيغة .h، و يكون اسم ملفنا الرأسي functions.h مثلا، و نكتب فيه المثال التالي:

```
1   /*Functions Header file*/
2   
3   int Add(int num1, int num2)
4   {
5   	return num1+num2;
6   }
7   
8   int Sub(int num1, int num2)
9   {
10  	return num1-num2;
11  }
12  
13  int Mul(int num1, int num2)
14  {
15  	return num1*num2;
16  }
17  
18  int Div(int num1, int num2)
19  {
20  	return num1/num2;
21  }
```

البرنامج ٢,٦,١: إنشاء ملف رأسي

ثم قم بإنشاء الملف الرأسي للمشروع في نفس المكان الذي قمت بإنشاء فيه الملف الرأسي functions.h، و قم بالكتابة فيه ما يلي:

```
1   #include<stdio.h>
2   #include"functions.h"
3   
4   main()
5   {
6   	int num1 = 30, num2 = 10;
7   
8   	printf("%d + %d = %d\n", num1, num2, Add(num1, num2));
9   	printf("%d - %d = %d\n", num1, num2, Sub(num1, num2));
10  	printf("%d * %d = %d\n", num1, num2, Mul(num1, num2));
11  	printf("%d / %d = %d\n", num1, num2, Div(num1, num2));
12  }
```

البرنامج ٢,٦,٢: ضم الملف الرأسي

في السطر الثاني قمنا بإضافة الملف الرأسي functions.h و لكن بطريقة مختلفة و هي وضع اسم الملف بين إقتباسين و ذلك لأن الملف الرأسي functions.h موجود في نفس المكان الذي موجود به الملف النصي الرئيسي main.c، أما إذا أردت كتابة:

```
1  #include<stdio.h>
2  #include<functions.h>
3
4  main()
5  {
6      int num1 = 30, num2 = 10;
7
8      printf("%d + %d = %d\n", num1, num2, Add(num1, num2));
9      printf("%d - %d = %d\n", num1, num2, Sub(num1, num2));
10     printf("%d * %d = %d\n", num1, num2, Mul(num1, num2));
11     printf("%d / %d = %d\n", num1, num2, Div(num1, num2));
12 }
```

البرنامج ٢,٦,٣: ضم ملف رأسي موجود بالمجلد include

فيجب عليك وضع الملف الرأسي functions.h في نفس المكان الذي موجود به الملف الرأسي stdio.h(أي في المجلد *include* الموجود في مجلد المترجم).

٢,٦,١ اسم الملف الرأسي:

يمكن كتابة أرقام في بداية اسم الملف الرأسي و أيضا يمكن استعمال مؤثرات الجمع و الطرح، و الرموز التي لا يمكن استعمالها في اسم الملف الرأسي هي: / ? : & \ * " > < | # %، و أقصى حد يمكن إعطاءه لاسم الملف الرأسي هو ٢٥٦ رمز.

٢,٦,٢ متى نستعمل الملفات الرأسية:

تستعمل الملفات الرأسية عند كتابة برامج كبيرة، مثلا إذا كنت تستعمل كثير دوال الرياضيات في برامج فيستحسن أن تقوم بكتابة ملف رأسي باسم math.h حيث تضع به جميع الدوال التي تريد استعمالها في المستقبل، و إذا كنت تستعمل دوال الرسم أيضا قم بإنشاء ملف رأسي لها باسم design.h أو graphics.h حيث تكون به أغلب دوال الرسم، و هكذا حتى تكون لديك مكتبة كبيرة خاصة بك.

٢,٦,٣ الأخطاء المحتملة:

١. عند إضافة ملف رأسي يجب التأكد أنه موجود في نفس مكان المشروع.

٢,٦,٤ تمارين:

١. أكتب ملف رأسي باسم math.h حيث به الدالة abs و التي تقوم بإخراج القيمة المطلقة للعدد المدخل، ثم قم باستعمال الدالة abs في برنامجك.

٢,٧ الإدخال و الإخراج في الملفات Files I/O

سنعرف في هذا الدرس طريقة التعامل مع الملفات في كل من الإدخال و الإخراج (قراءة الملفات، و إنشاء الملفات و كتابة عليها).

٢,٧,١ الإخراج في الملفات:

الإخراج يعني بناء برنامج يقوم بإخراج (إنشاء) ملفات ذات امتداد يقوم بتحديده المبرمج، حيث تحتوي تلك الملفات على بيانات. المثال الأول في هذا الدرس سيكون عبارة عن برنامج يطلب من المستخدم كتابة اسم الملف الــذي يريــد إنشاءه مع امتداده، و يطلب منه أيضا إدخال النص الذي يريد حفظه في الملف، المثال:

```
1   #include<stdio.h>
2
3   main()
4   {
5         FILE *File;
6         char FileName[255];
7         char String[255];
8
9         printf("Enter the name of file(with type)\n[MAX Character 255]: ");
10        gets(FileName);
11
12        printf("Creating File...\n");
13        File = fopen(FileName, "w");
14        printf("File Created.\n");
15
16        printf("TEXT:\n");
17        gets(String);
18
19        printf("Save Text to the file %s...\n", FileName);
20        fprintf(File, "%s", String);
21  }
```

البرنامج ٢,٧,١: طريقة فتح ملف

في السطر الخامس قمنا بإنشاء مؤشر للبنية FILE و ذلك لأن الدوال الخاصة بالملفات جميع تتطلب مؤشر للبنية FILE. في السطر السادس قمنا بالإعلان سلسلة حرفية و التي ستكون اسم الملف. و في السطر السابع سلسلة حرفيـــة و الـــتي ستكون النص الذي سنضعه في الملف. و في السطر العاشر طلبنا من المستخدم إدخال اسم الملف مع امتداده و أنه أقصى عدد الأحرف التي يمكن إدخالها هو ٢٥٥. و في السطر العاشر قمنا باستعمال الدالة gets بدل الدالة scanf لأخــذ اسم الملف، و سبب ذلك هو:

إن استعملنا الدالة scanf لإدخال اسم الملف فربما يقوم المستخدم بإدخال اسم مكون من كلمتين منفصلتين مثل *test test.txt* فهنا الدالة scanf ستتوقف عن الفراغ أي أن اسم الملف سيكون *test* و بدون امتداد، و هـذا هـو سـبب استعمال الدالة gets لأنها تأخذ الاسم كامل حتى و إن كانت فراغات. و في السطر الثالث عشر قمنا بإعطاء المؤشـر File عنوان الدالة fopen و التي هي مختصرة من *file open*، الدالة لها وسيطين، الوسيط الأول خاص باسم الملـف و الوسيط الثاني فهو النمط أي نوع استخدام الملف، و هنا استعملنا الكتابة لذا كتبنا الرمز w و الذي يعني *write*، و توجد أحرف أخرى سنعرفها فيما بعد. و في السطر السابع عشر ينتظر البرنامج من المستخدم لكي يقوم بإدخـال نـص و استعمالنا هو الدالة gets لنفس السبب السابق. و أخيرا السطر العشرين حيث توجد الدالة التي تقوم بإدخال النص إلى الملف، و الدالة هي fprintf و هي مثل الدالة printf و لكنها تتعامل مع الملفات أما printf فهي تتعامـل مـع الشاشة، الدالة fprintf مختصرة من *file print format* و لها وسيط إضافي على الدالة printf و هو الوسيط الأول و هو مؤشر الملف الذي نريد الكتابة فيه أما باقي الوسائط فهي نفسها وسائط الدالة printf و أيـضا نفـس طريقـة استعمالها. سنقوم الآن بكتابة المثال السابق باستخدام المؤشرات و الدوال و الملفات الرأسية (مع جعل البرنـامج أكثـر مرونة) كي نعتاد علي استعمالها. أولا نقوم بإنشاء ملف رأسي باسم fileio.h و نقوم بالكتابة عليه التالي:

```
1  /*fileio.h header file
2  for files functions*/
3  #include<stdlib.h> /*for exit() fonction*/
4  
5  void CreateFile(const char *FileName, /*for the name of file*/
6                  const char *String)   /*for text of file*/
7  {
8      FILE *FileOut;
9  
10     if(*FileName == '\0')
11     {
12         printf("Error in the name of file!\n");
13         exit(1);
14     }
15 
16     if((FileOut = fopen(FileName, "w"))==NULL){
17         printf("Can't create file!\n");
18         exit(1);
19     }else{
20         fprintf(FileOut, "%s", String);
21     }
22     fclose(FileOut);
23 }
```

البرنامج ٢,٧,٢: طريقة إنشاء ملف

هذا الملف هو المرحلة الأولى من البرنامج، و شرحه هو:

في السطر الثالث أضفنا الملف الرأسي stdlib.h و هو مختصر من *standard library*، و أضفناه لاستعمال الدالة exit (...) و التي تقوم بالخروج من البرنامج بدون تنفيذ باقي الأوامر. قمنا بالإعلان عن الدالة CreateFile في السطر الخامس مع وسيطين الأول لاسم الملف و الثاني لنص الملف. في السطر العاشر قمنا بوضع مقارنة بين اسم الملف الذي أدخله المستخدم و الصفر، و هو في حالة أن المستخدم لم يدخل أي حرف أو اسم للملف فسيتم الخروج من البرنامج لتجنب الأخطاء و تنبيه المستخدم عن وجود خطأ في اسم الملف. و في السطر السادس عشر قمنا بمقارنة أخرى و هي بين المؤشر FileOut و NULL حيث NULL هي:

```
#define NULL 0
```

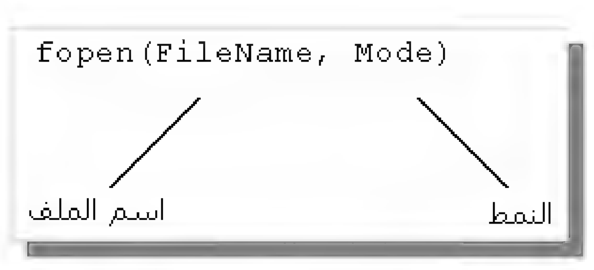
و يفضل كتابة NULL أحسن من الصفر، المقارنة هي إن كانت أخطاء مثل إدخال أحرف غير متفق عليها مثل <> في اسم الملف فهنا أيضا سيتم الخروج من البرنامج بدون إكمال تنفيذ باقي الأوامر، وفي حالة أن المقارنة خاطئة فسيتم كتابة النص إلى الملف. و استعملنا في السطر الثالث عشر و الثامن عشر الدالة exit حيث لها وسيط واحد و هو الوضع أما ١ و تعني صحيح *true* هنا سيتم الخروج من البرنامج، أو ٠ و تعني خطأ *false* و هنا لن يتم الخروج من البرنامج. و أخيرا السطر الثاني و العشرين و هي الدالة fclose و هي مختصرة من *file close* و التي تقوم بإغلاق الملف عند الانتهاء منه حتى يمكننا استعماله مرة أخرى ستلاحظ ذلك فيما بعد. هذا بالنسبة للملف الرأسي fileio.h، الآن نقوم بإنشاء الملف الرئيسي لمشروعنا باسم main.c:

```
1   #include<stdio.h>
2   #include"fileio.h"
3
4   main()
5   {
6       char FileName[255];
7       char String[255];
8
9       printf("Enter the name of file(with type)\n");
10      printf("[MAX Character 255]: ");
11      gets(FileName);
12
13      printf("TEXT:\n");
14      gets(String);
15
16      printf("Creating File and save text...\n");
17      CreateFile(FileName, String);
18  }
```

البرنامج ٢,٧,٣: إستعمال الدالة CreateFile من الملف الرأسي fileio.h

٢,٧,١,١ الدالة fopen:

هي من دوال الملف الرأسي stdio.h، و هي خاصة بالتعامل مع الملفات و لها وسيطين، الأول به سلسلة حرفية و هي اسم الملف، و الوسيط الثاني هو النمط أو الوضع الذي تريد استعمالها (القراءة أو الكتابة)، صورة توضيحية:



الشكل ٢,٧,١: طريقة فتح ملف

٢,٧,١,٢ الدالة fclose:

و هي أيضا من دوال الملف الرأسي stdio.h، و هي أيضا خاصة بالتعامل مع الملفات و لها وسيط واحد و هو مؤشر لـ FILE *، نكتب فيه اسم مؤشر البنية لكي يتم إغلاقها، و فائدة إغلاقها هي كي نستطيع قراءتها مرة أخرى في البرنامج.

٢,٧,١,٣ الدالة exit:

من دوال الملف الرأسي stdlib.h، لها وسيط واحد حيث يمكن أن يحمل القيمة ١ أو القيمة ٠، إذا كانت القيمة ١ تعني الخروج من البرنامج مباشرة بدون تنفيذ باقي الأوامر، أما إذا كانت ٠ فسيتم تجاهلها.

٢,٧,٢ إضافة نص في نهاية الملف:

الأمثل السابقة في هذا الدرس عندما تقوم بكتابة اسم ملف موجود سابقا فسيتم فقده، و هنا سنعرف كيفية نقوم بإضافة نص في أخر الملف بدون فقد البيانات السابقة. نفس الطريقة السابقة تمام فقط بدل الحرف w في الدالة fopen نكتب الحرف a، في السطر السادس عشر:

```
16 | if((FileOut = fopen(FileName, "w"))==NULL){
```

و يصبح على الشكل التالي:

```
16 | if((FileOut = fopen(FileName, "a"))==NULL){
```

و الحرف *a* يعني *appending*.

٢,٧,٣ الإدخال في الملفات:

الإدخال تعني القراءة، و في الملفات هي قراءة محتوى ملف و استعماله في البرنامج. في نفس الملف الرأسي الـسابق fileio.h قم بإضافة التالي:

```
1  void DisplayFile(const char *FileName)
2  {
3         FILE *FileIn;
4         char String[1024];
5  
6         if(*FileName == '\0')
7         {
8                printf("Error in name of file!\n");
9                exit(1);
10        }
11 
12        if((FileIn = fopen(FileName, "r"))==NULL){
13               printf("Can't Open file!\n");
14               exit(1);
15        }else{
16               fgets(String, 1024, FileIn);
17               printf("%s", String);
18        }
19        fclose(FileIn);
20 }
```

البرنامج ٢,٧,٤: إنشاء دالة تقوم بعرض محتوى ملف

هنا نكون قد أعلنا عن الدالة التي ستقوم بقراءة الملفات، و هي مشابه بدالة إنشاء الملفات. الدالة تحتوي علـى وسـيط واحد و هو سلسلة حرفية لاسم الملف المراد قراءته. في السطر الرابع قمنا بالإعلان عن مصفوفة و هناك نقوم بوضع نص الملف. في السطر الثاني عشر، في الدالة fopen استعملنا النمط *r* بدل *w*، الحرف *r* يعني *read*. في السطر السادس عشر استعملنا الدالة fgets، و هي من دوال الملف الرأسي stdio.h و هي مختصرة من *file get string*، لها ثلاثة وسـائط، الأول لاسم السلسلة الحرفية، الثاني لحجم السلسلة و الثالث لمؤشر ملف الإدخال FILE *FileIn. تقوم الدالة fgets بوضع سلسلة حروف الملف (بحجم الوسائط الثاني) في الوسائط الأول و الذي هو عبارة عن سلسلة حروف، ثم طبـع سلسلة حروف في السطر السابع عشر.

و الآن قم بإضافة الأوامر التالية في نهاية الدالة الرئيسية main() في الملف الرأسي الرئيسي main.c:

```
1  printf("/////////Reading\\\\\\\\\n");
2  DisplayFile(FileName);
```

و هنا استعملنا الدالة لاستعراض محتوى الملف.

٢,٧,٤ النمط +w و +a و +r:

درسنا سابقا كل من الأنماط *w* (للكتابة) و *a* (لإضافة نص في نهاية ملف) و *r* (لقراءة ملف)، و الآن سنرى نفس الأنماط السابقة مع إضافات و هي:

٢,٧,٤,١ النمط +w:

هنا يتم إنشاء ملف فارغ للقراءة و الكتابة معا، و إذا كان الملف موجود سابقا فسيتم فقد جميع محتوياته.

٢,٧,٤,٢ النمط +a:

هنا يتم إضافة نص في نهاية الملف إذا كان موجود و إن لم يكون موجود يتم إنشاءه، و أيضا يستعمل هذا النمط للقراءة.

٢,٧,٤,٣ النمط +r:

هذا النمط للقراءة و الكتاب و لكن يجب أن يكون الملف موجود.

و توجد أنمط أخرى و هي b، s، r، t و d، و لكهان غير مهمة، الأنماط السابقة هي المهمة و التي تستعمل بكثرة.

٢,٧,٥ دوال أخرى خاصة بالتعامل مع الملفات:

توجد دوال عديدة لتعامل مع الملفات و كلها شبيه بالتي قرأنها سابقا، و جميعها من دوال الملف الرأسي stdio.h، الدوال هي:

٢,٧,٥,١ الدالة fprintf و الدالة fscanf:

الدالة fprintf درسنها سابقا، أما الدالة fscanf فهي مكافئة للدالة scanf و هي مختصرة من *file scan file*، و لكنها هنا لا تأخذ قيم من المستخدم، بل تأخذها من قيم من ملف، أي أنها خاصة بالإدخال للملفات، مثال سريع:

```
1  #include<stdio.h>
2
3  main()
4  {
5      FILE *FileOut;
```

```
6       char Str[100];
7       int Var;
8
9       FileOut = fopen("fscanf.txt", "w+");
10
11      fprintf(FileOut, "%s %d", "Hello", 255);
12
13      fseek(FileOut, 0, SEEK_SET);
14
15      fscanf(FileOut, "%s", Str);
16      fscanf(FileOut, "%d", &Var);
17
18      printf("%s\n", Str);
19      printf("%d\n", Var);
20      fclose(FileOut);
21  }
```

البرنامج ٢,٧,٥: إستعمال الدالة fprintf و الدالة fscanf

في السطر الخامس قمنا بالإعلان عن مؤشر للبنية FILE باسم FileOut. في السطر السادس سلسلة حرفية بحجم ١٠٠، و هي التي ستحمل نص الملف. في السطر السابع متغير و هو الذي سيحمل القيمة الموجودة في الملف. في السطر التاسع قمنا بوضع اسم الملف الذي سنفتحه و نقرأ منه البيانات الموجودة. في السطر الحادي عشر وضعنا في الملف سلسلة حرفية و هي *Hello* و القيمة ٢٥٥. في السطر الثالث عشر توجد دالة و هي fseek و هي من دوال الملف الرأسي stdio.h و هي مختصرة من *file seek*، و هي تحرك مؤشر الملف إلى مكان يقوم بتحديده المبرمج، لها ثلاثة وسائط الأول هو اسم لمؤشر البنية FILE و الثاني هو عدد البايتات التي يبدأ منها الوسيط الثالث و غالبا ما تكون . لكي يتم قراءة جميع البيانات، و الوسيط الثالث هو وضع المؤشر و له ثلاثة ثوابت و هي:

١. SEEK_SET و هو وضع مؤشر الملف في البداية.
٢. SEEK_CUR و هو الموقع الحالي لمؤشر الملف.
٣. SEEK_END و هو وضع مؤشر الملف في نهايته.

و في مثالنا السابق استعمالنا SEEK_SET كي نبدأ بقراءة بيانات الملف من البداية و وضعنا القيمة . في الوسيط الثاني كي نقرأ نفحص جميع البيانات. و في السطر الرابع عشر قمنا بفحص سلسلة الحروف الموجود في الملف و نسخه في السلسلة الحرفية Str، و هو النص Hello. في السطر الخامس عشر قمنا بفحص العدد الصحيح ٢٥٥ و نسخه في المتغير Var. و في السطر الثامن عشر و التاسع عشر قمنا بطباعة النتائج.

٢,٧,٥,٢ الدالة fgets و fputs:

الدالة fgets درسنها سابقا، و الدالة fputs هي مكافئة للدالة puts و لكنها خاصة بالملفات و هي مختصرة من *file put string*، و هي تقوم بطباعة النصوص في الملفات و ليست مثل الدالة puts حيث تقوم بطباعة نص على الشاشة، مثل سريع حول الدالة fputs:

```
1  #include<stdio.h>
2
3  main()
4  {
5          FILE *FileOut;
6
7          FileOut = fopen("fputs.txt", "w");
8
9          fputs("Hello, World!\n", FileOut);
10     fclose(FileOut);
11 }
```

البرنامج ٢,٧,٦: إستعمال الدالة fputs

في السطر التاسع استعملنا الدالة fputs حيث الوسيط الأول هو النص المراد طبعه، و الوسيط الثاني هو اسم المؤشر FileOut حيث فيه سيتم طباعة النص.

٢,٧,٥,٣ الدالة fgetc و الدالة fputc:

الدالة fgetc تأخذ حرف واحد من ملف، و الدالة fputc تقوم بطباعة حرف واحد إلى ملف معين، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5          FILE *FileOut;
6          FILE *FileIn;
7          char ch;
8
9          FileOut = fopen("fputc.txt", "w");
10         FileIn = fopen("fputc.txt", "r");
11
12         fputc('A', FileOut);
13         fclose(FileOut);
14
15         ch = fgetc(FileIn);
16         fclose(FileIn);
17
18         printf("%c\n", ch);
19 }
```

البرنامج ٢,٧,٧: إستعمال الدالة fgetc و الدالة fputc

في السطر الثالث عشر إن لم تستعمل الدالة fclose فسترى نتائج غير مرغوب بها، و هنا ستعرف أهميتـــها. و يمكـــن كتابة المثال السابق على هذه الطريقة:

```
1  #include<stdio.h>
2  
3  main()
4  {
5  	FILE *FileInOut;
6  	char ch;
7  
8  	FileInOut = fopen("fputc.txt", "w");
9  
10 	fputc('A', FileInOut);
11 	fclose(FileInOut);
12 
13 	FileInOut = fopen("fputc.txt", "r");
14 
15 	ch = fgetc(FileInOut);
16 	fclose(FileInOut);
17 
18 	printf("%c\n", ch);
19 }
```

البرنامج ٢,٧,٨: إستعمال الدالة fgetc و الدالة fputc (٢)

هذه هي الدوال المهمة حاليا في العامل مع الملفات، توجد دوال أخرى كثيرة حول التعامل مع الملفات و سندرســـها في درس المكتبة القياسية للغة *C*.

٢,٧,٦ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

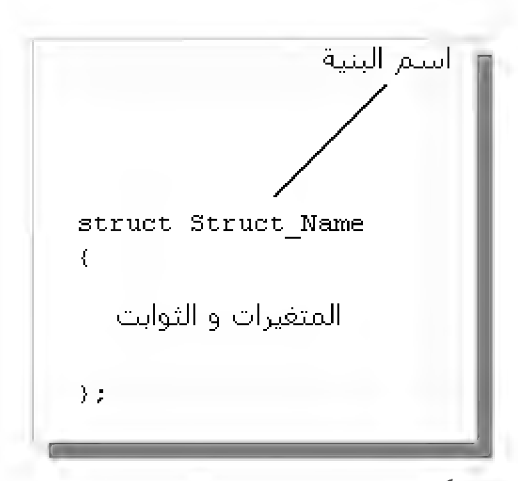
٢,٧,٧ تمارين:

١. أكتب برنامج يقوم بعمليات الجمع و الطرح الضرب و القسمة، و يتم حفظ النتـــائج في ملـــف *results.dat* تلقائيا.

٢,٨ التراكيب Structures

التراكيب (البنيات) هي مجموعة من متغير واحد أو أكثر تجمع تحت اسم واحد يسهل استعمالها، و المـــتغيرات في التراكيب ليس مثل المتغيرات في المصفوفات، يمكن أن تكون به متغيرات مختلفة الأنواع، و التراكيب يمكن أن تحمل أي نوع من متغيرات لغة C حتى مصفوفات أو مؤشرات أو تراكيب داخل تراكيب أخرى، و جميع المتغيرات الموجود داخل التراكيب تسمى بأعضاء لتراكيب.

للإعلان عن بنية نقوم بكتابة الكلمة المحجوزة struct ثم اسم البنية و نقوم بفتح حاضنة و نكتب المتغيرات و الثوابت التي نريدها (الأعضاء) ثم نغلق الحاضنة مع وضع فاصلة منقوطة، صورة توضيحية:



الشكل ٢,٨,١: طريقة الإعلان عن بنية

٢,٨,١ اسم البنية Struct Name:

اسم البنية له شروط مثل شروط اسم المتغير و هي:

- أن لا يتجاوز اسم البنية أكثر من ٣١ حرف.
- أن لا يبدأ اسم البنية بأرقام.
- أن لا يكون اسم البنية يحتوي على مؤثرات مثل الجمع و الطرح و....
- أن لا يكون اسم البنية يحتوي على رموز مثل % و # و { و...(باستثناء الرمز _).
- أن لا يكون اسم البنية مستعمل سابقا لاسم دالة أو متغير أو بنية أخرى.
- أن لا يكون اسم البنية من أحد أسماء الكلمات المحجوزة.

و الإعلان عن البنيات يستحسن أن يكون دائما خارج الدالة الرئيسية و قبلها، مثال:

```
1   #include<stdio.h>
2
3   struct _2D
4   {
5         int x;
6         int y;
7   };
8
9   main()
10  {
11        struct _2D Struct_2D;
12        int x, y;
13
14        printf("Enter X: ");
15        scanf("%d", &Struct_2D.x);
16
17        printf("Enter Y: ");
18        scanf("%d", &Struct_2D.y);
19
20        for(x=0;x<=Struct_2D.x;x++)
21        {
22              printf("%c", 219);
23              for(y=0;y<Struct_2D.y;y++)
24                    printf("%c", 219);
25              printf("\n");
26        }
27
28  }
```

البرنامج ٢,٨,١: طريقة الإعلان و إستعمال البنيات

هذا البرنامج يقوم برسم المربعات حسب القيم التي يقوم بإدخالها المستخدم. في السطر الثالث قمنا بالإعلان عن البنيـة _2D. و في السطر الخامس و السادس قمنا بالإعلان عن متغيرين x و y، حيث يعتبران عضوين للبنية _2D. في السـطر الحادي عشر قمنا بكتابة الكلمة المحجوزة struct مع اسم البنية السابقة _2D و الاسم Struct_2D و الـذي سـيتم العمل عليه في هذا البرنامج، و لا يمكن كتابة Struct_2D _2D بدون الكلمة المحجوزة struct، و لا يمكن اسـتعمال البنية _2D مباشرة لذا يجب أن نقوم بإعلان عن متغير للبنية _2D و التي هي Struct_2D في برنامجنا هذا. أمـا بـاقي السطور فهي مفهومة. و يمكن الإعلان عن أكثر من بنية، فمثلا المثال السابق في السطر الحادي عشر يمكننا كتابة:

```
struct _2D Struct_2D_1, Struct_2D_2;
```

و هنا يمكن استعمال كل من البنيتين Struct_2D_1 و البنية Struct_2D_2. أما إذا أردت تجاهـل كتابـة السـطر الحادي عشر، فيمكن ذلك و لكن يجب أن نعطي لبنيتنا معرف و سنكتبه قبل الفاصلة المنقطة:

```
1   struct _2D
```

```
2  {
3        int x;
4        int y;
5  }Struct_2D;
```

و في حالة نريد التعريف عن أكثر من اسم البنية فنفصل بين اسم و اسم بفاصلة مثل:

```
1  struct _2D
2  {
3        int x;
4        int y;
5  }Struct_2D_1, Struct2D_2;
```

و يصبح المثال على الشكل التالي:

```
1   #include<stdio.h>
2   
3   struct _2D
4   {
5         int x;
6         int y;
7   }Struct_2D;
8   
9   main()
10  {
11        int x, y;
12  
13        printf("Enter the position X: ");
14        scanf("%d", &Struct_2D.x);
15  
16        printf("Enter the position Y: ");
17        scanf("%d", &Struct_2D.y);
18  
19        for(x=0;x<=Struct_2D.x;x++)
20        {
21              printf("%c", 219);
22              for(y=0;y<Struct_2D.y;y++)
23                    printf("%c", 219);
24              printf("\n");
25        }
26  
27  }
```

البرنامج ٢,٨,٢: طريقة الإعلان و إستعمال البنيات (٢)

و توجد طرق أخرى لتعامل مع البنيات، مثلا يمكن إعطاء قيمة سابقة لمتغير في بنية و استعماله في البرنامج، مثال:

```
1  #include<stdio.h>
2  
3  struct _Value
```

```
4  {
5         int x;
6         float y;
7         char *Str;
8  }Value;
9
10 main()
11 {
12        Value.x = 10;
13        Value.y = 10.00;
14        Value.Str = "Hello, World";
15
16        printf("%d\n", Value.x);
17        printf("%f\n", Value.y);
18        printf("%s\n", Value.Str);
19 }
```

البرنامج ٢,٨,٣: طريقة الإعلان و إستعمال البنيات (٣)

أو يمكن إعطاء القيم مباشرة بعد التعرف عن اسم للبنية مثل:

```
1  #include<stdio.h>
2
3  struct _Value
4  {
5         int x;
6         float y;
7         char *Str;
8  }Value = {10, 10.00, "Hello, World"};
9
10 main()
11 {
12        printf("%d\n", Value.x);
13        printf("%f\n", Value.y);
14        printf("%s\n", Value.Str);
15 }
```

البرنامج ٢,٨,٤: إعطاء قيم لأعضاء بنية مباشرة بعد التعير عن إسم المبنية

و لا يمكن إعطاء قيمة عند الإعلان عن البنية مثل:

```
1  struct _Value
2  {
3         int x = 10;
4         float y = 10.00;
5         char *Str = "Hello World";
6  };
```

لأن في هذه الحالة لا معنى للبنية _Value و لا معنى لاستعمال البنية أصلا لأنه يمكن كتابة:

```
1  #include<stdio.h>
2
```

```
3   int x = 10;
4   float y = 10.00;
5   char *Str = "Hello World";
6
7   main()
8   {
9       printf("%d\n", x);
10      printf("%f\n", y);
11      printf("%s\n", Str);
12  }
```

البرنامج ٢,٨,٥: أفضل من إستعمال أعضاء البنية

٢,٨,٢ البنيات باستخدام الكلمة المحجوزة union:

يمكننا الإعلان عن البنيات باستعمال الكلمات المحجوزة union بنفس الطرق السابقة، و الفرق الوحيد بين استعمال البنية باستخدام الكلمة المحجوزة struct و الكلمة المحجوزة union هو عند اســتعمال البنيــات باســتخدام union فالنتائج لن تكون مثل البنيات التي بـــ struct مثلا أنظر إلى المثال التالي:

```
1   #include<stdio.h>
2
3   union _Union
4   {
5       int x;
6       int y;
7   }Union;
8
9   main()
10  {
11      Union.x = 10;
12      Union.y = 15;
13
14      printf("Union.x = %d\n", Union.x);
15      printf("Union.z = %d\n", Union.y);
16  }
```

البرنامج ٢,٨,٦: طريقة إستخدام بنية معرفة بـــ union

في هذا البرنامج بدل أن تكون النتائج ١٠ ١٥ فسيتكون ١٥ ١٥، و أيضا ستختلف النتائج إن استعملت التالي:

```
1   #include<stdio.h>
2
3   union _Union
4   {
5       int x;
6       int y;
7   }Union;
8
9   main()
10  {
11      Union.y = 15;
12      Union.x = 10;
13
```

```
14        printf("Union.x = %d\n", Union.x);
15        printf("Union.z = %d\n", Union.y);
16  }
```

البرنامج ٢,٨,٧: طريقة إستخدام بنية معرفة بـــ union (٢)

يمكنك الآن استنتاج الفرق بين البنيات باستخدام الكلمة المحجوزة struct و البنيات باســتخدام الكلمــة المحجــوزة union، و الذي هو إشتراك جميع المتغيرات في عنوان واحد، و إن غيرنا قيمة متغير واحدة فستكون تلك القيمة لجميع متغيرات البنية، مثال:

```
1   #include<stdio.h>
2
3   union _Union
4   {
5         int x1, x2;
6         int y1, y2;
7   }Union;
8
9   main()
10  {
11        Union.x1 = 10;
12
13        printf("Value of Union.x1 = %d\n", Union.x1);
14        printf("Value of Union.x2 = %d\n", Union.x2);
15        printf("Value of Union.y1 = %d\n", Union.y1);
16        printf("Value of Union.y2 = %d\n", Union.y2);
17
18        printf("Address of Union.x1 = %p\n", &Union.x1);
19        printf("Address of Union.x2 = %p\n", &Union.x2);
20        printf("Address of Union.y1 = %p\n", &Union.y1);
21        printf("Address of Union.y2 = %p\n", &Union.y2);
22  }
```

البرنامج ٢,٨,٨: طريقة إستخدام بنية معرفة بـــ union (٣)

و عند استعمال بنيات باستخدام الكلمة المحجوزة union بها متغيرات مختلفة الأنواع فستكون النتائج غير موثوقة، مثال:

```
1   #include<stdio.h>
2
3   union _Union
4   {
5         int x;
6         float y;
7   }Union;
8
9   main()
10  {
11        Union.x = 10;
12        Union.y = 15.00;
13
14        printf("Union.x = %d\n", Union.x);
15        printf("Union.z = %f\n", Union.y);
16  }
```

البرنامج ٢,٨,٩: طريقة إستخدام بنية معرفة بـــ union (٤)

بالنسبة لنتيجة متغير العدد الحقيقي y ستكون صحيحة، أما المتغير x فستكون نتيجتها غير مرغوبة، و يمكنك اســتنتاج السبب.

٢,٨,٣ المصفوفات و المؤشرات على البنيات:

لا أقصد بنية بها أعضاء من نوع مصفوفات أو مؤشرات، بل أقصد البنية نفسها، بالنسبة للمصفوفات مع البنية فهي شبيه بطريقة الإعلان عن مصفوفات طبيعية، مثال:

```
1  #include<stdio.h>
2
3  struct _Arr
4  {
5         int x;
6  }Arr[2];
7
8  main()
9  {
10        Arr[0].x = 10;
11        Arr[1].x = 20;
12
13        printf("Arr[0].x = %d\n", Arr[0].x);
14        printf("Arr[1].x = %d\n", Arr[1].x);
15 }
```

البرنامج ٢,٨,١١: المصفوفات على البنيات

و أيضا يمكن كتابة:

```
1  struct _Arr
2  {
3         int x;
4  }Arr[2] = {10, 20};
```

بدل:

```
1  Arr[0].x = 10;
2  Arr[1].x = 20;
```

و يمكن أيضا كتابة مصفوفة لبنية ثنائية أو ثلاثية الأبعاد أو أكثر بنفس الطرق السابقة التي درسنها في درس المصفوفات، و يمكن أيضا كتابة مصفوفة لبنية تحتوي على متغيرات لأنواع عديدة مثل:

```c
#include<stdio.h>

struct _Arr
{
	int x;
	float y;
	char *Str;
}Arr[2] = {
	{10, 10.00, "Str1"},
	{20, 20.00, "Str2"}
};

main()
{
	/*Arr[0] :*/
	printf("Arr[0].x = %d\n", Arr[0].x);
	printf("Arr[0].y = %f\n", Arr[0].y);
	printf("Arr[0].Str = %s\n", Arr[0].Str);

	/*Arr[1] :*/
	printf("Arr[1].x = %d\n", Arr[1].x);
	printf("Arr[1].y = %f\n", Arr[1].y);
	printf("Arr[1].Str = %s\n", Arr[1].Str);
}
```

البرنامج ٢,٨,١٢: المصفوفات على البنيات (٢)

و هذا مثال للمؤشرات:

```c
#include<stdio.h>

struct _ptr
{
	int x;
}Addr_ptr, *ptr;

main()
{
	ptr = &Addr_ptr;
	ptr->x = 10;

	printf("ptr->x = %d\n", ptr->x);

	/*Or*/
	(*ptr).x = 20;
	printf("(*ptr).x = %d\n", (*ptr).x);
}
```

البرنامج ٢,٨,١٣: المؤشرات على البنيات

طريقة استعمال مؤشر لبنية تختلف قليلا عن استعمال مؤشرات طبيعية، بالنسبة للإعلان فهي نفسها، أما إعطاء العنوان و القيمة تختلف. في السطر العاشر أعطينا للمؤشر ptr عنوانا و هو عنوان البنية Adrr_ptr. و في السطر الحادي عشر أعطينا للعضو x القيمة ١٠، و يتم إعطاء قيم لأعضاء مؤشر بنية عبر الرمزين -> ثم اسم العضو، ثم قيمته.

٢,٨,٤ إعلان بنية داخل بنية:

يمكن استعمال بنية داخل بنية، مثلا إذا أردنا أن نرسم خط مستقيم، هذا يحتاج إلى نقتطين، الأولى هي بداية المستقيم و الثانية هي نهاية المستقيم، و يجب أن يكون لكل نقطة مكانها على شاشة الحاسوب في كل من x و y، مثال:

```
1  #include<stdio.h>
2
3  struct _line
4  {
5      struct _point
6      {
7          int x;
8          int y;
9      }point_1, point_2;
10 }line;
11
12 main()
13 {
14     /*point 1:*/
15     line.point_1.x = 10;
16     line.point_1.y = 10;
17     /*point 2:*/
18     line.point_2.x = 100;
19     line.point_2.y = 10;
20 }
```

البرنامج ٢,٨,١٤: إعلان بنية داخل بنية

و هنا استعملنا البنية _point داخل البنية _line، و يجب أن نقوم بالتعرف لأسماء _point مباشرة عند كتابتها كـي نستطيع استعمال المتغيرين x و y.

٢,٨,٥ الأخطاء المحتملة:

١. لا يمكن استعمال بنية مباشرة مثل:

```
1  #include<stdio.h>
2
3  struct Struct
4  {
5      int x, y;
6  };
7
```

```
8  main()
9  {
10       Struct.x = 10;
11       Struct.y = 20;
12 }
```

البرنامج ٢,٨,١٥: الخطأ ١

٢. لا يمكن الإعلان عن بنيتين من نفس الاسم.

٢,٨,٦ تمارين:

١. أكتب بنية بسم time بها ثلاثة أعضاء و هي hh, mm, ss و كلها من النوع int، و نطلب من المستخدم إدخال الساعة و الدقيقة و الثانية الحالية و نعطي تلك القيم للبنية time ثم نطبع الساعة على الطريقة *HH :MM :SS*.

٢,٩ ملخص للفصل الثاني، معا إضافات

درسنا سابقا دالة الإدخال scanf، و قلنا أنه يجب أن نعطيه اسم متغير مع مؤثر العنوان &، و ذلك يعني أنه عند كتابة:

```
1  #include<stdio.h>
2
3  main()
4  {
5         int i;
6
7         scanf("%d", &i);
8  }
```

البرنامج ٢,٩,١: الدالة scanf

فهنا سيتم وضع القيمة المدخلة في عنوان المتغير i، لأن الدالة scanf تتعامل مع المؤشرات، و يمكن كتابة المتغير بـــدون مؤثر، و لكن النتائج لن تكون صحيحة.

٢,٩,١ معنى دالة بها وسيط void:

أحيانا ما تجد في بعض الأمثل دوال بها وسيط void مثل:

```
1   #include<stdio.h>
2
3   Function(void);
4
5   main()
6   {
7
8   }
9
10  Function(void)
11  {
12
13  }
```

البرنامج ٢,٩,٢: معنى دالة بها وسيط void

أو تجد الدالة الرئيسية نفسها بها هذا الوسيط، مثال:

```
1  main(void)
2  {
3
4  }
```

البرنامج ٢,٩,٣: معنى دالة بها وسيط void (٢)

مثل هذه الدوال الوسيط الخاص بها لا يعني شيء، و كلمة void إنجليزية و هي تعني فراغ، و هذا هو معنى كتابة دوال بها وسيط من هذا النوع، يعني التأكيد على أن الدالة فارغة الوسائط، و مثل تلك الدوال يمكن تجاهل كتابة الكلمة المحجوزة void، حيث أن كلا من هذه الأمثلة:

```
1  #include<stdio.h>
2  
3  Function();
4  
5  main()
6  {
7  
8  }
9  
10 Function()
11 {
12 
13 }
```

البرنامج ٢,٩,٤: معنى دالة بها وسيط void (٣)

و المثال:

```
1  main()
2  {
3  
4  }
```

البرنامج ٢,٩,٥: معنى دالة بها وسيط void (٣)

مكافئة للأمثلة السابقة.

٢,٩,٢ الكلمة المحجوزة static:

كلمة static تعني ساكن و هي تستعمل مع المتغيرات حيث تجعلها ثابت بطرقة ستفهمها من هذا المثال:

```
1  #include<stdio.h>
2  
3  main()
4  {
5        int x;
6  
7        for(x=0; x<10; x++)
8        {
9              static int Static = 0;
10             Static++;
11 
12             printf("%d\n", Static);
13       }
```

```
14 | }
```

البرنامج ٢,٩,٦: طريقة إستعمال الكلمة المحجوزة static

في السطر التاسع استعملنا الكلمة المحجوزة static مع المتغير Static داخل حلقة و أعطيناه القيمة ٠، و استعملنا مؤشر الزيادة في السطر العاشر مع نفس المتغير Static، و في السطر الثاني عشر قمنا بطباعة النتائج. إذا قمنا بإزالة الكلمة المحجوزة static من المتغير Static فستكون النتائج ثابتة و هي ١ عشرة مرات، أما عند استعمالها فسترى أن النتيجة غير ثابتة و تعمل بتزايد. و تستعمل الكلمة المحجوزة static بكثرة في الدوال، مثال:

```
1  #include<stdio.h>
2
3  int Test_static(int a);
4
5  main()
6  {
7       int i;
8
9       for(i=0;i<=10;i++)
10          printf("%d * %d = %d\n", i, i, Test_static(i));
11 }
12
13 int Test_static(int a)
14 {
15      static int c = 0;
16      int a_c = a*c;
17      c++;
18
19      return a_c;
20 }
```

البرنامج ٢,٩,٧: طريقة إستعمال الكلمة المحجوزة static (٢)

و في حالة إزالة الكلمة المحجوزة static فستكون جميع النتائج ٠. إن لم نقم بتحديد قيمة لمتغير static فستكون قيمته ٠ تلقائيا، و ليس مثل المتغيرات الطبيعية التي تنتج قيم عشوائية.

٢,٩,٣ الكلمة المحجوزة typedef:

تستعمل الكلمة المحجوزة typedef مع كل من المتغيرات و البنيات، و هي تعطي إمكانيات الإعلان عنها مثل البنية، مثال:

```
1  #include<stdio.h>
2
3  typedef int Decimal;
4
5  main()
6  {
7       /* or typedef int Decimal; here */
```

```
8         Decimal Var_1, Var_2;
9         Decimal Var_3;
10
11        Var_1 = 10;
12        Var_2 = 20;
13        Var_3 = Var_1 + Var_2;
14
15        printf("%d\n", Var_3);
16 }
```

البرنامج ٢,٩,٨: طريقة إستعمال الكلمة المحجوزة typedef

في السطر الثالث قمنا بالإعلان عن متغير من نوع int مع الكلمة المحجوزة typedef باسم Decimal. في السطر الثامن و التاسع استعملنا المتغير Decimal للإعلان عن المتغيرات Var_1 و Var_2 و Var_3 و تعاملنا معا تلك المتغيرات كأي كتغيرات أخرى. و في مثالنا هذا استبدلنا الكلمة المحجوزة int بـ Decimal حيث حجمها سيكون حجم النوع الذي تم الإعلان به، و لا يمكن إعطاءها قيم. أما بالنسبة للمبنية فهي نفسها مشابه مثلا:

```
1  #include<stdio.h>
2
3  struct _2D
4  {
5        int x, y;
6  };
7
8  main()
9  {
10       struct _2D A;
11       struct _2D B;
12
13       A.x = 10;
14       B.y = 20;
15
16       printf("%d\n", A.x);
17       printf("%d\n", B.y);
18 }
```

البرنامج ٢,٩,٩: طريقة إستعمال الكلمة المحجوزة typedef (٢)

و لكنها تجعلها أكثر مرونة من السابقة، مثال:

```
1  #include<stdio.h>
2
3  typedef struct _2D
4  {
5        int x, y;
6  }Struct1, Struct2;
7
8  main()
9  {
10       Struct1 S1_1, S1_2;
11       Struct2 S2_1, S2_2;
12
```

```
13 	S1_1.x = 10, S1_2.x = 20;
14 	S2_1.y = 30, S2_2.y = 40;
15 }
```

البرنامج ٢,٩,١٠: طريقة إستعمال الكلمة المحجوزة typedef (٣)

ملاحظة: عند استعمال متغيرات أو دوال أو بنيات مع الكلمة المحجوزة typdef فلا يمكن إعطاءها قيم سابقة مثل:

```
1 #include<stdio.h>
2 
3 main()
4 {
5 	typedef float PI;
6 	PI = 10.00;
7 	printf("%f\n", PI);
8 }
```

البرنامج ٢,٩,١١: طريقة إستعمال الكلمة المحجوزة typedef (٤)

٢,٩,٤ برامج تدريبية:

في هذا الجزء من الدرس سنرى بعض البرامج التي ستساعدك على فهم لغة *C* بشكل مبسط، مع شرح سريع لكل برنامج:

٢,٩,٤,١ البرنامج الأول، النسخ:

في هذا البرنامج نقوم بكتابة دالة تقوم بنسخ سلسلة حروف إلى سلسلة حروف أخرى فارغة، المثال:

الملف str.h:

```
1  /*string.h
2  
3  strcopy(pointers)
4  */
5  void strcopy(char *From, char *To){
6  	while((*To++ = *From++)!='\0')
7  		;
8  }
9  
10 /*strcopy(arrays);
11 void strcopy(char From[], char To[]){
12 	int i;
13 	i = 0;
14 
15 	while((To[i] = From[i])!='\0')
16 		++i;
17 }*/
```

البرنامج ٢,٩,١٢: النسخ، الملف str.h

الملف الرئيسي:

```
/*main.c*/

#include<stdio.h>
#include"str.h"

main()
{
	char *From = "STRING";
	char Empty[6];

	strcopy(From, Empty);

	printf(Empty);
}
```

البرنامج ٢,٩,١٣: النسخ، الملف الرئيسي

هنا الدالة ستقوم بنسخ ما هو موجود في السلسلة From إلى السلسلة Empty، ثم نطبع محتوى السلسلة Empty كي نتأكد من النتائج.

٢,٩,٤,٢ تبادل قيم بين وسيطين:

في هذا البرنامج نقوم بإنشاء دالة تقوم بوضع قيمة المتغير الوسيط الأول في الوسيط الثاني و قيمة الوسيط الثاني في الوسيط الأول، المثال:

```
#include<stdio.h>

void Change(int *a, int *b)
{
	int c;
	c = *a;
	*a = *b;
	*b = c;
}

main()
{
	int a, b;
	a = 5;
	b = 10;

	printf("a = %d, b = %d\n", a, b);

	Change(&a, &b);

	printf("a = %d, b = %d\n", a, b);
}
```

البرنامج ٢,٩,١٤: تبادل قيم بين وسيطين

إذا استعملنا متغيرات في مكان المؤشرين b* ,a* فإن النتائج البرنامج ستكون خاطئة، و هنا نرى فائدة التعامـــل مـــع عناوين الذاكرة.

٢,٩,٤,٣ التغير في قيم ثوابت:

قلنا سابقا أنه لا يمكن التغير في قيم ثوابت، و لكن عبر المؤشرات يمكننا ذلك، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5         const int Const = 10;
6         int *ptr = &Const;
7
8         printf("Const = %d\n", Const);
9         *ptr = 5;
10        printf("Const = %d\n", Const);
11
12 }
```

البرنامج ٢,٩,١٥: تغير قيمة ثابت

٢,٩,٤,٤ عكس سلسلة نصية:

في هذا البرنامج نقوم بإنشاء دالة بها وسيط لسلسلة من حروف، حيث تقوم تلك الدالة بعكس تلـــك السلـــسلة في نفسها، المثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  void Reverse_Str(char *);
5
6  main()
7  {
8         char *str = "Hello, World!";
9
10        printf("%s\n", str);
11        Reverse_Str(str);
12        printf("%s\n", str);
13 }
14
15 void Reverse_Str(char *String){
16        int i = strlen(String)-1, j = 0;
17        char ch;
18
19        while(j<i){
20              ch = String[j];
```

```
21              String[j] = String[i];
22              String[i] = ch;
23              j++, i--;
24          }
25  }
```

البرنامج ٢,٩,١٦: عكس سلسلة نصية

٢,٩,٤,٥ التحويل من النظام العشري إلى النظام الثنائي:

من خلال هذا البرنامج يمكنك استنتاج كيفية التحويل إلى باقي الأنظمة، المثال:

```
1   #include<stdio.h>
2
3   void ConvertToBinary(int);
4
5   main()
6   {
7       int Decimal;
8
9       printf("Decimal Number: ");
10      scanf("%d", &Decimal);
11      printf("%d in Binary = ", Decimal);
12      ConvertToBinary(Decimal);
13  }
14
15  void ConvertToBinary(int num){
16      int i = 0, Binary[32];
17
18      while(num>0){
19          if((num%2)==0){
20              Binary[i] = 0;
21              num /= 2, ++i;
22          }
23          else
24          {
25              Binary[i] = 1;
26              num /= 2, ++i;
27          }
28      }
29
30      --i;
31
32      while(i>=0)
33      {
34          printf("%d", Binary[i]), --i;
35
36      }
37
38      printf("\n");
39  }
```

البرنامج ٢,٩,١٧: التحويل من النظام العشري إلى النظام الثنائي

٢,٩,٤,٦ التحويل من الحروف الصغيرة إلى الحروف الكبيرة:

في هذا البرنامج نقوم بإنشاء دالة تقوم بتحويل سلسلة حرفية من الحروف الصغيرة إلى الحروف الكبيرة، المثال:

```
1  #include<stdio.h>
2  
3  void To_Capital_letter(char ch[]);
4  
5  main()
6  {
7        char *ch = "hello";
8  
9        printf("Small Letters: %s\n", ch);
10       To_Capital_letter(ch);
11       printf("Capital Letters: %s\n", ch);
12 }
13 
14 void To_Capital_letter(char ch[])
15 {
16       int i=0;
17 
18       while(ch[i]!='\0'){
19             if(ch[i]>=97 && ch[i]<=122)
20                   ch[i] = ch[i]-32;
21             ++i;
22       }
23 }
```

البرنامج ٢,٩,١٨: التحويل من الحروف الصغيرة إلى الحروف الكبيرة

و يمكن أيضا استعمال العكس، من الأحرف الكبيرة إلى الأحرف الصغيرة، استنتج ذلك.

٢,٩,٥ الدالة wcscpy و الدالة wcsncpy:

الدالة wcscpy مكافئة للدالة strcpy و هي من دوال الملف الرأسي string.h و اسمها مختصر من *wide character string copy*، تقوم بنفس عمل الدالة strcpy فقط هي للأحرف العريضة، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3  
4  main()
5  {
6        wchar_t *wStr = L"Hello";
8        wchar_t Empty[20];
9  
10       wcscpy(Empty, wStr);
11 
12       wprintf(L"%s\n", Empty);
13 }
```

البرنامج ٢,٩,١٩: طريقة إستعمال الدالة wcscpy

و أيضا الدالة wcsncpy مكافئة للدالة strncpy و هي من دوال الملف الرأسي string.h، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main()
5  {
6        wchar_t *wStr = L"Hello";
7        wchar_t Empty[20];
8
9        wcsncpy(Empty, wStr, 4);
10
11       wprintf(L"%s\n", Empty);
12 }
```

البرنامج ٢,٩,٢٠: طريقة إستعمال الدالة wcsncpy

٢,٩,٦ الدالة wcscat و الدالة wcsncat:

و كلا من الدالتين wcscat و wcsncat مكافئة للدالتين strcat و strncat، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main()
5  {
6        wchar_t *wStr = L"Hello";
7        wchar_t *wStr2 = L", World!";
8
9        wcscat(wStr, wStr2);
10
11       wprintf(L"%s\n", wStr);
12
13       /*
14       wcsncat(wStr, wStr2, 4);
15
16       wprintf(L"%s\n", wStr);
17       */
18 }
```

البرنامج ٢,٩,٢١: طريقة إستعمال الدالة wcscat و الدالة wcsncat

٢,٩,٧ الدالة getwchar و putwchar:

الدالة getwchar مكافئة لدالة getchar، و هي من دوال الملف الرأسي stdio.h، واسمها مختصر مـــن *get wide character*، و أيضا الدالة putwchar مكافئة لدالة putchar، و مختصرة من *put wide character*، مثال:

```
1  #include<stdio.h>
2
3  main()
4  {
5        wchar_t wch;
6
7        wch = getwchar();
```

```
8        putwchar(wch);
9   }
```

البرنامج ٢,٩,٢٢: طريقة إستعمال الدالة getwchar و الدالة putwchar

٢,٩,٨ الدالة _getws و _putws:

الدالة _getws مكافئة لدالة gets، و الدالة _putws مكافئة لدالة puts، و هما من دوال الملف الرأسي stdio.h، و كل من الحرفين الإضافيين ws مختصرين من *wide string*، مثال:

```
1   #include<stdio.h>
2
3   main()
4   {
5        wchar_t wch[255];
6
7        _getws(wch);
8        _putws(wch);
9   }
```

البرنامج ٢,٩,٢٣: طريقة إستعمال الدالة _getws و الدالة _putws

٢,٩,٩ جدول ASCII (صورة):

كلمة *ASCII* مختصرة من *American Standard Code for Information Interchange*، هو جدول لمجموعة من الرموز مستندة على الأبجدية الرومانية تمثل رموز في الحاسبات ، الرموز على الجدول التالي:

إذا أردت استعمل أي رمز من الرموز السابقة قم بكتابة التالي:

```
1   #include<stdio.h>
2
3   main()
4   {
5        printf("The number %d is the character %c in ASCII code.\n", 210, 210);
6   }
```

البرنامج ٢,٩,٢٤: طباعة حرف عبر رقمه في جدول أسكي

و هنا سيتم طبع الرمز رقم ٢١٠ من جدول *ASCII* و هو الرمز ╥، و يمكن اختيار أي رمز من الرموز السابقة فقط نكتب الرقم و نطبعه على شكل حرف (رمز).

```
0   :     1   :☺    2   :☻    3   :♥    4   :♦    5   :♣    6   :♠    7   :     8   :     9  :
10  :     11  :     12  :     13  :     14  :♫    15  :☼    16  :►    17  :◄    18  :↕    19:‼
                                                                  20  :¶    21  :§    22:▬
23  :↨    24  :↑    25  :↓    26  :→    27  :←    28  :∟    29  :↔    30  :▲    31  :▼    32:
33  :!    34  :"    35  :#    36  :$    37  :%    38  :&    39  :'    40  :(    41  :)    42:*
43  :+    44  :,    45  :-    46  :.    47  :/    48  :0    49  :1    50  :2    51  :3    52:4
53  :5    54  :6    55  :7    56  :8    57  :9    58  ::    59  :;    60  :<    61  :=    62:>
63  :?    64  :@    65  :A    66  :B    67  :C    68  :D    69  :E    70  :F    71  :G    72:H
73  :I    74  :J    75  :K    76  :L    77  :M    78  :N    79  :O    80  :P    81  :Q    82:R
83  :S    84  :T    85  :U    86  :V    87  :W    88  :X    89  :Y    90  :Z    91  :[    92:\
93  :]    94  :^    95  :     96  :`    97  :a    98  :b    99  :c    100:d     101:e     102:f
103:g     104:h     105:i     106:j     107:k     108:l     109:m     110:n     111:o     112:p
113:q     114:r     115:s     116:t     117:u     118:v     119:w     120:x     121:y     122:z
123:{     124:|     125:}     126:~     127:⌂     128:Ç     129:ü     130:é     131:â     132:ä
133:à     134:å     135:ç     136:ê     137:ë     138:è     139:ï     140:î     141:ì     142:Ä
143:Å     144:É     145:æ     146:Æ     147:ô     148:ö     149:ò     150:û     151:ù     152:ÿ
153:Ö     154:Ü     155:¢     156:£     157:¥     158:₧     159:ƒ     160:á     161:í     162:ó
163:ú     164:ñ     165:Ñ     166:ª     167:º     168:¿     169:⌐     170:¬     171:½     172:¼
173:¡     174:«     175:»     176:░     177:▒     178:▓     179:│     180:┤     181:╡     182:╢
183:╖     184:╕     185:╣     186:║     187:╗     188:╝     189:╜     190:╛     191:┐     192:└
193:┴     194:┬     195:├     196:─     197:┼     198:╞     199:╟     200:╚     201:╔     202:╩
203:╦     204:╠     205:═     206:╬     207:╧     208:╨     209:╤     210:╥     211:╙     212:╘
213:╒     214:╓     215:╫     216:╪     217:┘     218:┌     219:█     220:▄     221:▌     222:▐
223:▀     224:α     225:ß     226:Γ     227:π     228:Σ     229:σ     230:µ     231:τ     232:Φ
233:Θ     234:Ω     235:δ     236:∞     237:φ     238:ε     239:∩     240:≡     241:±     242:≥
243:≤     244:⌠     245:⌡     246:÷     247:≈     248:°     249:∙     250:·     251:√     252:ⁿ
253:²     254:■     255:
```

الشكل ٢,٩,١: جدول أسكي

٢,٩,١٠ معلومات أكثر حول المتغيرات:

توجد متغيرات خارجية و متغيرات محلية، الأولى هي متغيرات عامة يمكن استعمالها بصفة عامة، أما المتغيرات المحلية فهي متغيرات لها حدودها.

٢,٩,١٠,١ المتغيرات المحلية:

هي متغيرات يمكن استعمالها في الدالة التي تم الإعلان عنها، مثال:

```
1   #include<stdio.h>
2
3   void Func()
4   {
5           int a;       /*Local Variable*/
6   }
7
8   main()
9   {
10          int b;       /*Local Variable*/
11  }
```

البرنامج ٢,٩,٢٥: المتغيرات المحلية

هنا كل من المتغير a في الدالة Func و المتغير b في الدالة الرئيسية يعتبرا متغيرات محلية حيث لا يمكن استعمال المتغير a في الدالة الرئيسية لأنه خاص بالدالة Func، و لا يمكن استعمال المتغير b في الدالة Func لأنه معرف في الدالة main، و هذا هو مفهوم المتغيرات المحلية.

٢,٩,١٠,٢ المتغيرات الخارجية (العامة):

هي متغيرات يمكن التعامل معها في جميع الدوال، أي في البرنامج بأكمله، و يتم الإعلان عنها خارج جميع الدوال، مثال:

```
1  #include<stdio.h>
2  
3  int ab;                  /*External Variable*/
4  
5  void Func()
6  {
7        ab = 10;     /*Use The External Variable ab*/
8  }
9  
10 main()
11 {
12       Func();
13       printf("%d\n", ab); /*print The External Variable ab*/
14 }
```

البرنامج ٢,٩,٢٦: المتغيرات الخارجية

المتغير ab هو المتغير العام للبرنامج، حيث يمكن التعامل معه في جميع الدوال، أعطين للمتغير ab قيمة في الدالة Func ثم قنا بتنفيذ الدالة Func في الدالة الرئيسية ثم طبع محتوى المتغير الخارجي ab في السطر الثالث عشر. إذا لم تكن للمتغيرات العامة قيم سابقة فستكون قيمها ٠ تلقائيا.

٢,٩,١٠,٣ الكلمة المحجوزة extern:

تستعمل الكلمة المحجوزة extern مع المتغيرات داخل دوال، و تستعمل لجعل متغير محلي مشترك مع متغير خارجي، مثلا يمكننا الإعلان عن متغير محلي و متغير خارجي بنفس الاسم و لكن بقيم مختلفة، مثال:

```
1  #include<stdio.h>
2  
3  int ab;
4  
5  void Func()
6  {
7        int ab;
8        ab = 10;
```

```
9        printf("%d\n", ab);
10  }
11
12  main()
13  {
14        ab = 5;
15        Func();
16        printf("%d\n", ab);
17  }
```

البرنامج ٢,٩,٢٧: طريقة إستعمال الكلمة المحجوزة extern

في هذا المثال المتغير المحلي ab الموجود داخل الدالة Func لا علاقة له مع المتغير الخارجي ab، أما إذا أردنـــا أن نجعـــل قيمتهما مشترك نضيف الكلمة المحجوزة extern إلى المتغير المحلي، مثال:

```
1   #include<stdio.h>
2
3   int ab;
4
5   void Func()
6   {
7         extern int ab;
8         ab = 10;
9         printf("%d\n", ab);
10  }
11
12  main()
13  {
14        ab = 5;
15        Func();
16        printf("%d\n", ab);
17  }
```

البرنامج ٢,٩,٢٨: طريقة إستعمال الكلمة المحجوزة extern (٢)

هنا ستكون قيمة المتغير الخارجي ab هي نفس قيمة المتغير الداخلي ab.

٢,٩,١٠,٤ الكلمة المحجوزة auto:

تستعمل الكلمة المحجوزة auto مع المتغيرات، و هي تعني *automatic* أي آلي، و تستعمل مع المتغيرات لتبين أن تلك المتغيرات افتراضية أي طبيعية و ليست ساكنة static، مثال:

```
1   #include<stdio.h>
2
3   void Func()
4   {
5         static int Static = 0;
6         auto int Auto = 0;
7
8         printf("Static = %d\n", Static++);
9         printf("Auto   = %d\n", Auto++);
```

```
10 }
11
12 main()
13 {
14     int i = 0;
15
16     while(i<=3){
17         Func();
18         i++;
19     }
20 }
```

البرنامج ٢,٩,٢٩: طريقة إستعمال الكلمة المحجوزة auto

و يمكن كتابة المتغير Auto بدون استخدام الكلمة المحجوزة auto.

٢,٩,١٠,٥ الكلمة المحجوزة register:

أيضا تستعمل مع المتغيرات العددية، و لا يمكن استعمالها مع مصفوفات أو بنيات أو متغيرات خارجية أو متغيرات ساكنة، استعمال متغيرات بها الكلمة المحجوزة register تعني وضع تلك المتغيرات في سجل الحاسوب، و سجل الحاسوب موجود في وحدة المعالجة المركزية *CPU (Central Processing Unit)*، مثال:

```
1 #include<stdio.h>
2
3 main()
4 {
5     register Reg_Var = 4;
6
7     printf("Reg_Var = %d\n", Reg_Var);
8 }
```

البرنامج ٢,٩,٣٠: طريقة إستعمال الكلمة المحجوزة register

و لا يمكن استعمال الإدخال لمتغيرات السجل.

٢,٩,١١ الكلمة المحجوزة sizeof:

تستعمل الكلمة المحجوزة sizeof لمعرفة أحجام البيانات، و منها يمكن معرفة الحجم الكامل المستعمل لبرامج، تكون النتيجة بالبايت *bytes*، مثال لمعرفة أحجام أنواع المتغيرات:

```
1 #include<stdio.h>
2
3
4 main()
5 {
6     int SizeOfArray[800];
7
```

```
8          printf("short       = %d Byte(s)\n", sizeof(short));
9          printf("int     = %d Byte(s)\n", sizeof(int));
10         printf("unsigned    = %d Byte(s)\n", sizeof(unsigned));
11         printf("signed      = %d Byte(s)\n", sizeof(signed));
12         printf("long        = %d Byte(s)\n", sizeof(long));
13         printf("float       = %d Byte(s)\n", sizeof(float));
14         printf("double      = %d Byte(s)\n", sizeof(double));
15         printf("SizeOfArray = %d Byte(s)\n", sizeof(SizeOfArray));
16  }
```

البرنامج ٢,٩,٣١: طريقة إستعمال الكلمة المحجوزة sizeof

وهنا ستتعرف على أحجام أنواع المتغيرات التي تريدها، و يمكن أيضا معرفة حجم مصفوفة كما في المثال.

٢,٩,١٢ استدعاء دالة لنفسها:

يمكننا إضافة هذه الطريقة إلى التكرار، حيث نستدعي دالة من نفسها، إن لم تكون شروط في الدالة فستكون الدالـة بلا نهاية، مثال:

```
1   #include <stdio.h>
2
3   void Func(int num)
4   {
5          printf("%d\n", num);
6          Func(num+1);
7   }
8
9   main()
10  {
11         Func(5);
12  }
```

البرنامج ٢,٩,٣٢: استدعاء دالة لنفسها

هنا استدعينا الدالة لنفسها في السطر السادس، و في هذا البرنامج سيتم التكرار إلى أن يصل إلى الحد الأقصى من القـيم التي يمكن أن يحملها نوع المتغير، و ها سيتوقف البرنامج عند الرقم ٦٥٥٣٥ لأنه العدد الأقصى الذي يمكن أن يحملـه المتغير int num، لنجعل الدالة تقوم بتكرار محدود نستعمل شرط مثل التالي:

```
1   #include <stdio.h>
2
3   void Func(int num)
4   {
5          if(num <= 10)
6          {
7                 printf("%d\n", num);
8                 Func(num+1);
9          }
10  }
11
```

```
12 main()
13 {
14       Func(5);
15 }
```

البرنامج ٢,٩,٣٣: استدعاء دالة لنفسها (٢)

هنا سيقوم البرنامج بالتكرار من العدد ٥ إلى العدد ١٠.

٢,٩,١٣ التحكم في طباعة النتائج:

في الدالة printf، يمكننا التحكم في طريقة طبع النتائج، سواء كانت النتائج عبارة عن أرقام أو حروف، فمثلا إذا كان لدينا عدد حقيقي له أربعة أرقام وراء الفاصلة و لا نريد طباعة إلا إثنين منها نستعمل الطرق التالية:

```
1  #include<stdio.h>
2
3  main(int argc, char *argv[]){
4        char str[] = "Hello, World!";
5        float flt  = 3.1415F;
6
7
8        /*String*/
9        printf("%s\n", str);
10       printf("%5s\n", str);
11       printf("%-5s\n", str);
12       printf("%5.5s\n", str);
13       printf("%-5.5s\n", str);
14
15       printf("\n");
16
17       /*Float number*/
18       printf("%10.0f\n", flt);
19       printf("%.1f\n", flt);
20       printf("%.2f\n", flt);
21       printf("%.3f\n", flt);
22       printf("%.4f\n", flt);
23 }
```

البرنامج ٢,٩,٣٤: طريقة التحكم في طباعة النتائج

جرب كل واحدة من الطرق السابقة حتى تستنتج كل طريقة و فائدتها.

٢,٩,١٤ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

٢,٩,١٥ تمارين:

١. أكتب برنامج يقوم بالتحويل من النظام العشري إلى النظام السداسي عشر.

3
الفصل الثالث – التقدم في لغة C

مقدمة: نهاية المطاف، هنا سننهي طريقنا في لغة *C*، لم يبقى إلى أشياء صغيرة (و لا أقول بسيطة) لها الفصل، و مـــا جعله يكون طويلا هو المكتبة القياسية لكثرة دوالها.

بالتوفيق إن شاء الله

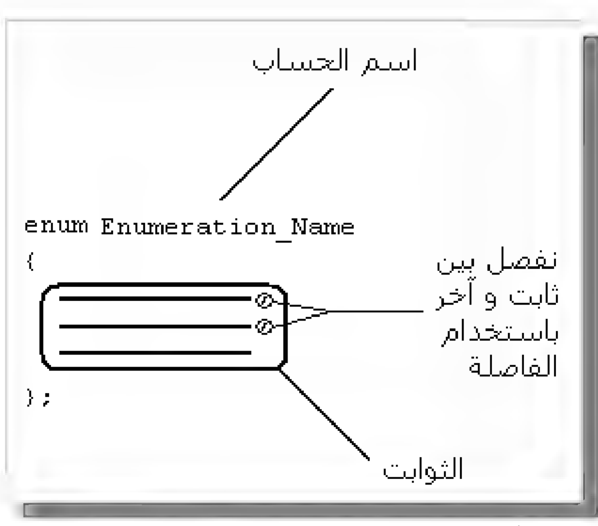
قال الله تعالى:

﴿ قل هل يستوي الذين يعلمون و الذين لا يعلمون ﴾

صدق الله تعالى

٣,١ الحساب Enumeration

الحساب (*Enumeration*) هو مجموعة من ثوابت أعداد صحيحة `int`، و يتم الإعلان عنها باستخدام الكلمة المحجوزة `enum` و التي هي مختصرة من *enumeration*، و هي مشابها للبنية في الإعلان عنها و طريقة التعامل معها. و هذه صورة توضيحية لطريقة استعمال الحسابات:



الشكل ٣,١,١: طريقة الإعلان عن الحساب

٣,١,١ اسم الحساب Enumeration Name:

لاسم الحساب شروط هي نفسها الشروط السابقة للمتغيرات، البنيات و الدوال، و الشروط هي:

- أن لا يتجاوز اسم الحساب أكثر من ٣١ حرف.
- أن لا يبدأ اسم الحساب بأرقام.
- أن لا يكون اسم الحساب يحتوي على مؤثرات مثل الجمع و الطرح و....
- أن لا يكون اسم الحساب يحتوي على رموز مثل % و # و { و...(باستثناء الرمز _).
- أن لا يكون اسم الحساب مستعمل سابقا لاسم دالة، متغير، بنية أو حساب آخر.
- أن لا يكون اسم الحساب من أحد أسماء الكلمات المحجوزة.

٣,١,٢ ثوابت الحساب:

ثوابت الحساب يمكن أن تحميل قيم أعداد صحيحة فقط، و يتم الفصل بين ثوابت و آخر باستعمال الفاصلة. إذا لم نحدد لتلك الثوابت قيم فسيتم إعطاءها قيم إفتراضية متسلسلة تبدأ من الصفر للثابت الأول ثم تتزايد حسب عدد الثوابت. سنقوم الآن بكتابة أبسط مثال لكيفية استعمال الكلمة المحجوزة enum و هو:

```
1  #include<stdio.h>
2
3  enum _COLOR
4  {
5         RED = 0,
6         BLUE,
7         GREEN
8  };
9
10 main()
11 {
12        enum _COLOR Color;
13
14        /*Color = RED;*/
15        Color = BLUE;
16        /*Color = GREEN;*/
17
18        if(Color == RED){
19               printf("Color = %d\n", Color);
20        }else if(Color == BLUE){
21               printf("Color = %d\n", Color);
22        }else if(Color == GREEN){
23               printf("Color = %d\n", Color);
24        }else
25               printf("Error!\n");
26 }
```

البرنامج ٣,١,١: طريقة إستعمال enum

و يمكن أيضا كتابة المثال بالطريقة التالية:

```
1  #include<stdio.h>
2
3  enum _COLOR
4  {
5         RED = 0,
6         BLUE,
7         GREEN
8  }Color;
9
10 main()
11 {
12        /*Color = RED;*/
13        Color = BLUE;
14        /*Color = GREEN;*/
15
16        if(Color == RED){
17               printf("Color = %d\n", Color);
18        }else if(Color == BLUE){
```

```
19 |                printf("Color = %d\n", Color);
20 |        }else if(Color == GREEN){
21 |                printf("Color = %d\n", Color);
22 |        }else
23 |                printf("Error!\n");
24 | }
```

البرنامج ٣,١,٢: طريقة إستعمال enum (٢)

و يمكن أيضا إعطاء القيمة المختار مباشرة عند الإعلان عن معرف للحساب _COLOR مثل كتابة:

```
enum _COLOR Color = BLUE;
```

أو كتابة:

```
enum _COLOR
{
        RED = 0,
        BLUE,
        GREEN
}Color = BLUE;
```

سنشرح الآن المثال السابق مع القليل من التفاصيل. قلنا سابقا أنه عند استعمال الكلمة المحجوزة enum لها قيم فهذا يعني أن تلك القيم ثابتة و لا يمكن التغير فيها، و يمكن تسمية أعضاء الكلمة المحجوزة enum بثوابت. و ثوابت الكلمة المحجوزة enum نقوم بكتابة أسماءها بدون كتابة نوعها، حيث قلنا سابقا أن نوعها هو أعداد صحيحة int.

أعطينا لثابت RED القيمة *0* و هذا يعني أن الثابت BLUE يساوي ١ و الثابت GREEN يساوي ٢. و يمكن أن لا نعطي للثابت RED القيمة ،، لأنه يتم إعطاءه القيمة ، تلقائيا إن لم تكن لديه قيمة سابقة.

أما إذا أعطينا لثابت BLUE القيمة ، فهنا ستصبح جميع الثوابت السابقة تحمل القيمة ، إن لم تكن لديها قيم، و هذا يعني أن الثابت RED سيحمل القيمة ، و الثابت BLUE أيضا يحمل القيمة ، أما الثابت GREEN فسيحمل القيمة ١. و يمكن أن نعطي أكثر من معرف للحسابات مثل البنيات، مثال:

```
1  | #include<stdio.h>
2  |
3  | enum _COLOR
4  | {
5  |        RED = 0,
6  |        BLUE,
7  |        GREEN
8  | };
9  |
10 | main()
11 | {
```

```
12         enum _COLOR cRed = RED,
13               cBlue = BLUE,
14               cGreen = GREEN;
15  }
```

البرنامج ٣,١,٣: طريقة إستعمال enum (٣)

أو الإعلان عنها مباشرة بعد الحساب مثل:

```
1   #include<stdio.h>
2
3   enum _COLOR
4   {
5         RED = 0,
6         BLUE,
7         GREEN
8   }cRed = RED, cBlue = BLUE, cGreen = GREEN;
9
10  main()
11  {
12
13  }
```

البرنامج ٣,١,٤: طريقة إستعمال enum (٤)

و توجد طرق أخرى كثيرة يمكن استعمالها مع الحسابات منها إعطاء لمتغير قيمة حساب مثل:

```
1   #include<stdio.h>
2
3   enum _NUMBER
4   {
5         Num = 10,
6         Num2 = 20
7   };
8
9   main()
10  {
11        enum _NUMBER Number = Num;
12        int i = 0;
13
14        printf("i      = %d\n", i);
15        i = Number;
16
17        printf("Number = %d\n", Number);
18        printf("i      = %d\n", i);
19  }
```

البرنامج ٣,١,٥: طريقة إستعمال enum (٥)

و يمكن أيضا استعمال الإدخال على الحساب مثل:

```
1   #include<stdio.h>
2
3   enum _COLOR
```

```
4   {
5         BLACK,
6         WHITE,
7         RED,
8         GREEN,
9         BLUE
10  };
11
12  main()
13  {
14        enum _COLOR Color;
15
16        printf("colors:\n");
17        printf("0)BLACK\n1)WHITE\n2)RED\n3)GREEN\n4)BLUE\n");
18        printf("choose a color:");
19        scanf("%d", &Color);
20
21        if(Color == BLACK){
22              printf("Your choice are Black\n");
23        }else if(Color == WHITE){
24              printf("Your choice are White\n");
25        }else if(Color == RED){
26              printf("Your choice are Red\n");
27        }else if(Color == GREEN){
28              printf("Your choice are Green\n");
29        }else if(Color == BLUE){
30              printf("Your choice are Blue\n");
31        }
32  }
```

البرنامج ٣,١,٦: طريقة إستعمال enum (٦)

٣,١,٣ الأخطاء المحتملة:

١. لا يمكن استعمال اسم الحساب مباشرة مثل:

```
1   #include<stdio.h>
2
3   enum _NUMBER
4   {
5         Zero,
6         One
7   };
8
9   main()
10  {
11        enum _NUMBER = ZERO;
12  }
```

البرنامج ٣,١,٧: الخطأ ١

٢. لا يمكن الإعلان عن حسابين من نفس الاسم.

٣,١,٤ تمارين:

١. أكتب برنامج به حساب باسم POWER_ به ثابتين، الأول باسم Off و الثاني باسم On، ثم قم باستخدام معرفة للحساب POWER_ باسم Power و استعملها للإدخال مباشرة باستخدام الدالة scanf، فإذا كانت النتيجة المدخل ٠ أي Off فسيتم الخروج من البرنامج، أما إذا كانت العكس أي ١ (On) فسيبقى البرنامج يستمر حتى تصبح القيمة المدخلة للحساب Power هي ٠.

٣,٢ وسائط الدالة الرئيسية Command-line Arguments

وسائط الدالة الرئيسية مختلفة عن غيرها من الوسائط، حيث يمكن إعتبارها كوسائط للبرنامج. و مثال على ذلك الأوامر التي يتم إستدعاءها في نظام *DOS* مثلا، بعض من هذه الأوامر تحتاج إلى وسائط مثل الأمر *del* و الـــذي يقـــوم بحذف ملف، و من وسائطه هي إسم الملف الذي سيتم حذفه، و بعض الوسائط خيارية مثل الخيار */p* و */s* مثلا.

٣,٢,١ الوسيط الأول لدالة الرئيسية:

أو بما يعرف بـــ *argc* إختصار لـــ *argument count* أي عداد، و هو معرف على شكل متغير من نـــوع الأعـــداد الصحيحة، و في هذا الوسيط يكون به عدد وسائط البرنامج التي تم إدخالها، و تكون قيمة هذا المتغير ١ و هي لإســـم البرنامج. و مثال على ذلك نأخذ المثال السابق و الذي هو الأمر *del*، و نكتب مثلا *del filename.ext /p*، هنا ستكون قيمة المتغير *argc* هي ثلاثة، الأولى هي إسم و مسار البرنامج *del*، و الثانية هي اسم الملف *filename.ext*، و القيمة الثالثة للخيار */p*، و من هذا المثال نستنتج أن المتغير *argc* يقوم بحساب وسائط البرنامج بالإعتماد على الفراغات (أي الفـــصل بين خيار و آخر بمساحة). مثال تطبيقي:

```
1  #include<stdio.h>
2
3  main(int argc)
4  {
5        printf("Program have %d argument(s).\n", argc);
6  }
```

البرنامج ٣,٢,١: الوسيط الأولى لدالة الرئيسية

و هنا قم بحفظ البرنامج باسم *test.c* ثم قم بإنشاء الملف التنفيذي له و من منفذ الأوامر قم بتنفيذ البرنامج على الشكل:

```
test this_is_the_first_argument and_this_is_the_second_argument
```

و ناتج هذه البرنامج يكون على الشكل:

```
Program have 3 argument(s).
```

ثلاثة وسائط، لأنه و كما قلنا أن اسم و مسار البرنامج أيضا يتم إضافته، و إذا كنت تريد عدد الوسائط فقط فيمكنك كتابة `argc-1`. و هنا المتغير `argc` يمكن إعطاءه إسم آخر، لأن المترجم يأخذها على شكل وسائط و ليس أسمـــاء، و لكن بعض المترجمات تأخذ هذه الوسائط على شكل أسماء، أي أنه لا يمكن التغير في الإسم `argc`.

٣,٢,٢ الوسيط الثاني لدالة الرئيسية:

إلى الآن عرفنا عدد الوسائط المدخل لبرنامجنا، و لكن كيف يمكننا أن نرى تلك الوسائط، أي القيم التي بها، ذلـــك يكون عبر الوسيط الثاني لدالة الرئيسية أي بما يعرف بـــ argv إختصار لـــ *argument vector*، حيث يكون عبارة عن مصفوفات، كل مصفوفة تأخذ وسيط، مثال:

```
1   #include<stdio.h>
2
3   main(int argc, char *argv[])
4   {
5         int i;
6
7         printf("Program have %d argument(s).\n", argc);
8
9         for(i=0;i<argc;i++)
10              printf("argv[%d]= %s\n", i, argv[i]);
11
12  }
```

البرنامج ٣,٢,٢: الوسيط الثاني لدالة الرئيسية

مثل المثال السابق، نقوم بتنفيذ البرنامج على الشكل:

```
test this_is_the_first_argument and_this_is_the_second_argument
```

و ناتج البرنامج يكون على الشكل التالي:

```
Program have 3 argument(s):
argv[0] = C:\...\...\TEST.EXE
argv[1] = this_is_the_first_argument
argv[2] = this_is_the_second_argument
```

هذه مجرد أمثلة لتبين طريقة عمل هذه الوسائط لا أكثر، أم طريقة استعمالها و الإعتماد عليها فهذا مثال شبه كامل:

```
1   #include<stdio.h>
2   #include<conio.h>
3
4   main(int argc, char *argv[])
5   {
6       char ch;
7       if(argc<2){
8                 printf("Delete one or more files.\n\n");
9                 printf("Usage: DEL FILENAME\n");
10      }
11      else{
```

```
12                 printf("Delete %s file, are you sure? (Y/N): ", argv[1]);
13                 if((ch=getche())=='y' || ch=='Y'){
14                     printf("\nDeleting %s...\n", argv[1]);
15                     remove(argv[1]);
16                     printf("%s deleted.\n", argv[1]);
17                 }
18         }
19 }
```

البرنامج ٣,٢,٣: برنامج ذات وسائط يقوم بحذف ملفات

في السطر السابع قمنا بعمل شرط و هو إن كان عدد وسائط البرنامج أقل من ٢، أي أن الوسائط المطلوبة غير متوفرة، ثم نظهر للمستخدم طريقة استعمال البرنامج. إذا كان عدد وسائط البرنامج أكثر من إثنين فإننا نقوم بعملية الحذف حيث يكون اسم الملف المراد حذفه في السلسلة `argv[1]` أي بعد سلسلة اسم و مسار البرنامج (`argv[0]`).

٣,٢,٣ الأخطاء المحتملة:

١. تأكد دائما أن المتغير `argc` تحمل القيمة ١ على الأقل، و هي لإسم و مسار البرنامج.

٣,٢,٤ تمارين:

لا توجد تمارين في هذا الدرس.

٣,٣ التوجيهات Directives(Preprocessor)

التوجيهات هي عبارة عن تعليمات خاص بالمترجم يتم تنفيذها قبل البدأ في الترجمة، و يمكن تميزها دائما بالرمز # في بداية اسم كل تعليمة، و هي كثيرة و سنتحدث عن كل منها مع البعض من التفاصيل.

٣,٣,١ التوجيه #include:

يعتبر هذا التوجيه من أهم التوجهات، و قد قمنا باستعمله و دراسته سابقا، و هو يطلب من المترجم بضم محتويات ملفات رأسية إلى مشروعنا، حيث يمكن استعمل دوال و ثوابت تلك الملفات الرأسية منها مباشرة بدون إعادة كتابتها.

و طريقة استعملها طريقتين هما:

الطريقة الأولى هي ضم ملفات رأسية خاصة بالمترجم، و غالبا ما تكون في مجلد باسم *include*، في هذه الحالة نكتب:

```
#include<FileName.h>
```

و هنا نكتب اسم الملف الذي نريد ضمه إلى مشروعنا في مكان FileName. الطريقة الثانية هي ضم ملف رأسي موجود في نفس المجلد الموجود به مشروعنا، و طريقة ضم الملفات في هذه الحالة تكون كالتالي:

```
#include"FileName.h"
```

و الفرق الوحيد بين الطريقتين هو كتابة الرمزين <> عندما نريد إضافة ملف رأسي من المجلد المترجم (*include*)، و كتابة الإقتباسين " " في حالة أن الملف الرأسي موجود في نفس المجلد المشروع. و يمكن أن نقول أن التوجيه #include خاص بربط الملفات ببعضها البعض.

٣,٣,٢ التوجيه #define:

يستعمل التوجيه #define في الإعلان الثوابت و التي تحدثنا عنها سابقا، و أيضا يستعمل في الإعلان عن المختصرات *Macros*، بالنسبة للإعلان عن الثوابت باستخدام التوجيه #define فهي كتالي:

```
#define Constant_Name Constant_Value
```

في مكان Constant_Name نقوم بكتابة اسم الثابت، و في مكان Constant_Value نكتب قيمة الثابت. و أما عـــن نوع القيمة المعطاة للثابت فإن التوجيه #define يقوم بتحديدها تلقائيا حيث يمكن استعمل أي نوع من الأنواع، مثال:

```
1 #define String          "Hello"
2 #define Character       'H'
3 #define fNumber         3.14
4 #define iNumber          10
```

أما بالنسبة للمختصرات *Macros* فهي مشابه للدوال، و طريقة استعملها كتالي:

```
#define Macro_Add(i, j) i+j
```

هنا قمنا بإنشاء مختصر به وسيطين، (و أيضا هنا لا نقوم بتحديد نوع تلك الثوابت) و قيمته هي الجمع بين الثابت i و الثابت j، مثال كامل:

```
1  #include<stdio.h>
2
3  #define Macro_Add(i, j) i+j
4
5  main(){
6        int l;
7
8        l = Macro_Add(10, 20);
9
10       printf("%d\n", l);
11
12       /*or*/
13
14       printf("%d\n", Macro_Add(10, 20));
15 }
```

البرنامج ٣,٣,١: مختصر يقوم بعملية جمع

٣,٣,٣ التوجيه #undef:

التوجيه #undef هو معاكس لتوجيه #define، حيث تقوم بإلغاء الثوابت و المختصرات التي نريد أن نقوم بإلغاءها و إعادة تعرفيها، و لكن في حالة الثوابت فيجب أن تلك الثوابت معلنة باستخدام التوجيه #define، مثال:

```
1 #include<stdio.h>
2
3 #define Constant        5
4 #define Macro_Add(i, j) i+j
```

```
5
6  #undef Constant
7  #undef Macro_Add
8
9  main(){
10         printf("%d\n", Constant); /*Error: 'Constant':
11                                       undeclared identifier*/
12
13         Macro_Add(5, 10);          /*Errors*/
14 }
```

البرنامج ٣,٣,٢: طريقة إستعمال التوجيه #undef

في السطر الثالث قمنا بالإعلان عن الثابت Constant و أعطيناه القيمة ٥، و في السطر الرابع قمنا بالإعلان عن المختصر Macro_Add، ثم قمنا بإلغاء كل من الثابت و المختصر في السطر السادس و السطر السابع، و عند استعمال الثابت Constant أو المختصر Macro_Add بعد إلغاءهما فسينبهك المترجم عن وجود أخطاء.

٣,٣,٤ التوجيهات #if، #elif، #else و #endif:

هي توجيهات تستعمل لعمليات شرطية قبل الترجمة، و طريقة استعملها هي نفس الطريقة معا كل من if و else، فهنا التوجيه #if مشابه لشرط if، و التوجيه #elif مشابه لاستعمال if else معا، و أخير التوجيه #else و هو أيضا مشابه لشرط else. و التوجيه #endif يعني نهاية الشروط، مثال:

```
1  #include<stdio.h>
2
3  #define dNum      20
4
5  #if dNum == 10
6          #define Result "dNum = 10"
7  #elif dNum == 20
8          #define Result "dNum = 20"
9  #else
10         #define Result "dNum = ??"
11 #endif
12
13 main(){
14         printf("%s\n", Result);
15 }
```

البرنامج ٣,٣,٣: طريقة إستعمال التوجيهات #if، #elif، #else و #endif

٣,٣,٥ التوجيه #ifdef و التوجيه #ifndef:

يستعمل التوجيه #ifdef في حالة أننا أردنا أنرى نرى إن كان هناك ثابت أو مختصر معرف سابق، مثال:

```
1  #include<stdio.h>
2
3  #define dNum
```

```
4
5  #ifdef dNum
6         #define Result "dNum Defined"
7  #else
8         #define Result "dNum Undefined"
9  #endif
10
11 main(){
12        printf("%s\n", Result);
13 }
```

البرنامج ٣,٣,٤: طريقة إستعمال التوجيه #ifdef

هنا ستكون نتيجة البرنامج هي dNum Defined، لأننا قمنا بالإعلان عن الثابت dNum سابقا، و يمكن كتابة المثال السابق بهذه الطريقة:

```
1  #include<stdio.h>
2
3  #define dNum
4
5  #if defined dNum
6         #define Result "dNum Defined"
7  #else
8         #define Result "dNum Undefined"
9  #endif
10
11 main(int argc, char *argv[]){
12        printf("%s\n", Result);
13 }
```

البرنامج ٣,٣,٥: طريقة أخرى مكافئة لـ #ifdef

و التوجيه #ifndef معاكس لتوجيه #ifdef، مثال:

```
1  #include<stdio.h>
2
3  #define dNum
4
5  #ifndef dNum
6         #define Result "dNum Undefined"
7  #else
8         #define Result "dNum Defined"
9  #endif
10
11 main(){
12        printf("%s\n", Result);
13 }
```

البرنامج ٣,٣,٦: طريقة إستعمال التوجيه #ifndef

و يمكن أيضا كتابة هذا مثال بالطريقة التالي:

```
1  #include<stdio.h>
2
3  #define dNum
4
5  #if !defined dNum
6          #define Result "dNum Undefined"
7  #else
8          #define Result "dNum Defined"
9  #endif
10
11 main(){
12         printf("%s\n", Result);
13 }
```

البرنامج ٣,٣,٧: طريقة أخرى مكافئة لـــ #ifndef

٣,٣,٦ التوجيه #line:

يستعمل التوجيه #line لتحديد سطر للملف الحالي أو سطر لملف غير الحالي، و ذلك غير مهم إلا في عملية بنـــاء المترجمات، و طريقة استعمله كالتالي:

```
#line LineNumber
```

هذه الطريقة في حالة أردنا تحديد رقم سطر للملف الحالي، حيث نكتب رقم السطر في مكان LineNumber، و الطريقة الأخرى هي:

```
#line LineNumber "FileName"
```

و عند استعمال هذه الطريقة يجب دائما كتابة اسم الملف دالة الاقتباسين " ". ستفهم طريقة استعمال هذا التوجيه أكثر عندما تعرف المختصرات المعرفة (سندرسها فيما بعد).

٣,٣,٧ التوجيه #error:

يساعد هذا التوجيه في تنبيه المبرمج في مرحلة الترجمة مباشرة، لأخطاء يقوم بتجهيزها المبرمج في حالة الوقوع فيهـــا، مثال:

```
1  #ifndef dNum
2  #error dNum are not declared!
3  #endif
4
5  main(){
6
7  }
```

البرنامج ٣,٣,٨: طريقة إستعمال التوجيه #error

هنا البرنامج لن يعمل و الخطأ هو الرسالة المكتبة بعد التوجيه #error، أي أن الثابت dNum غير معرف سابقا.

٣,٣,٨ التوجيه #pragma:

يستعمل التوجيه #pragma لتحكم في المترجم حسب رغبة المبرمج، و هي تختلف من مترجم لآخر، يستحسن مراجعة المساعد الخاص بالمترجم الذي تستعمله.

٣,٣,٩ الأسماء المعرفة Predefined Names:

هي مجموعة من المختصرات *Macros* جاهزة، و يمكن تميزها بالرمزين *Underscore* في بداية و نهاية اسم المختصر، و هذا جول لجميع الأسماء المعرفة:

المختصر	الشرح
__LINE__	ثابت عشري يحمل رقم سطر المصدر الحالي
__FILE__	سلسلة حرفية تحمل اسم الملف الجاري ترجمته
__TIME__	سلسلة حرفية تحميل الوقت الذي تم فيه الترجم
__DATE__	سلسلة حرفية تحميل التاريخ الذي
__STDC__	تكون قيمة هذا الثابت ١ إذا كان المترجم المستعمل مترجم قياسي للغة *C*.

الجدول ٣,٣,١: الأسماء المعرفة

مثال حول طريقة استعمال تلك الأسماء:

```
1  #include<stdio.h>
2
3  main(){
4      printf("Line: %d\n", __LINE__);
5      printf("File: %s\n", __FILE__);
6      printf("Date Of Compilation: %s\n", __DATE__);
7      printf("Time Of Compilation: %s\n", __TIME__);
8      printf("ANSI C(0=false, 1=true): %d\n", __STDC__);
9  }
```

البرنامج ٣,٣,٩: إستعمال الأسماء المعرفة

٣,٣,١٠ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

٣,٣,١١ تمارين:

لا توجد تمارين في هذا الدرس.

٣,٤ دوال ذات وسائط غير محددة

درسنا سابقا الدوال، و قلنا أنه يمكن عمل لها وسائط، و عند استعمال تلك الدوال يجـب أولا أن نـوفر لهـا الوسائط التي تريدها، فمثلا لا يمكن عمل مثلما هو موضح في هذا المثال:

```
1  #include<stdio.h>
2
3  int Add(int a, int b);
4
5  main(){
6         int a = 10, b = 20, c = 30, d;
7
8         d = Add(a, b, c);
9
10        printf("%d\n", d);
11 }
12
13 int Add(int a, int b){
14        return a+b;
15 }
```

البرنامج ٣,٤,١: توفير ثلاثة وسائط لدالة لها وسيطين

هنا البرنامج لن يعمل، و الخطأ في السطر الثامن و هو أن الدالة Add تحتوي على وسائط فقط، أما نحن فقد مررنا لهـا ثلاثة وسائط. و ستستنتج أن الدالتين printf و الدالة scanf لهما وسائط غير محددة و يمكن أن نزيد عليها وسـائط حسب رغباتنا، و سنقوم الآن بعمل استنساخ صغير لدالة printf و جعلها أكثر مرونة من السابقة:

```
1  #include<stdio.h>
2  #include<stdarg.h>
3
4  void NewPrintf(char *, ...);
5
6  main(){
7         NewPrintf("%d\a, %d\a, %d\a\n", 1, 2, 3);
8                NewPrintf("Hello!\n");
9  }
10
11 void NewPrintf(char *format, ...){
12        va_list Argument;
13        char *str;
14        int Decimal;
15
16        va_start(Argument, format);
17
18        for(str=format;*str;str++){
19               if(*str != '%'){
20                      putchar(*str);
21                      continue;
22               }
23
```

```
24              switch(*++str){
25                    case 'd':
26                    case 'i':
27                    case 'D':
28                    case 'I':
29                          Decimal = va_arg(Argument, int);
30                          printf("%d", Decimal);
31                          break;
32
33                    default:
34                          putchar(*str);
35                          break;
36              }
37        }
38
39        va_end(Argument);
40  }
```

البرنامج ٣,٤,٢: طريقة الإعلان دالة ذات وسائط غير محددة

أولا يجب ضم الملف الرأسي stdarg.h، و اسم الملف مختصر من *standard argument*، و هو الملف الذي سيعطينا الإمكانيات لجعل الدوال لها وسائط غير محددة. في السطر الرابع قمنا بالإعلان عن الدالة الجديدة لطبعة باسم NewPrintf، حيث لها وسيط واحد و هو الوسيط الرئيسي، أما تلك النقاط ... فهي تعني أنه ستتم تمرير لدالة عدد من الوسائط غير محددة. في السطر الثاني عشر قمنا بالإعلان عن مؤشر لـ va_list، و هو مختصرة من *variable list*، و هو من مؤشرات الملف الرأسي stdarg.h، حيث سيحمل هذا المؤشر عدد وسائط الدالة NewPrintf. في السطر السادس عشر قمنا باستعمال المختصر va_start، حيث له وسيطين، الوسيط الأول هو مؤشر لـ va_list، و الوسيط الثانية هو تمرير الوسيط الأول لدالة NewPrintf، و هنا سنقوم بتمرير المؤشر format. في السطر التاسع و العشرين قمنا باستعمال المختصر va_arg، و هو مختصر من *variable argument*، و له وسيطين، الوسيط الأول مؤشر لـ va_list، و الوسيط الثاني هو نوع المتغير الذي سيتم أخذ قيمته من مكان الوسيط الإضافي ثم نقله إلى المتغير Decimal، ثم طبعه مباشرة. و هذا مثال لكيفية جعل المثال الأول يعمل بشكل صحيح، المثال:

```
1   #include<stdio.h>
2   #include<stdarg.h>
3
4   int Add(int NumOfArg, ...);
5
6   main(){
7         int a = 10, b = 20, c = 30, d;
8
9         d = Add(3, a, b, c);
10
11        printf("%d\n", d);
12  }
13
14  int Add(int NumOfArg, ...){
15        va_list Arg;
16        int sum = 0, i;
```

```
17
18        va_start(Arg, NumOfArg);
19
20        for(i=0;i<NumOfArg;i++)
21              sum += va_arg(Arg, int);
22
23        return sum;
24  }
```

البرنامج ٣,٤,٣: طريقة الإعلان دالة ذات وسائط غير محددة (٢)

٣,٤,١ الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

٣,٤,٢ تمارين:

لا توجد تمارين في هذا الدرس.

٣,٥ المكتبة القياسية Standard Library

في هذا الدرس سنتعرف على أهم ثوابت، مختصرات و دوال لغة *C* القياسية، و كل مجموعة منـــها موجـــودة في ملف رأسي ذات اسم يدل على دورها.

٣,٥,١ الملف الرأسي assert.h:

يحتوي هذا الملف الرأسي على دالة و هي assert، بها وسيط واحد، و هي دالة تقوم بالمقارنة بين قيمـــتين حيـــث تكون النتيجة إما صفر أي خاطئة، أو غير الصفر أي صحيحة، إن كانت النتيجة خاطئة أي . فسيتم استعداء تلك الدالة كمختصر *macro*، حيث يقوم ذلك المختصرة بطباعة الخطأ مع اسم الملف الموجود به الخطأ و رقم السطر أثناء تـــشغيل البرنامج، و هنا سيتوقف البرنامج، أما إن كانت النتيجة غير الصفر فسيتم تجاهل المختصر assert، و لن تقوم الدالـــة assert بعمل شيء، مثال:

```
1  #include<stdio.h>
2  #include<assert.h>
3
4  main(){
5        assert(10<20); /*true*/
6        assert(20<10); /*false*/
7  }
```

البرنامج ٣,٥,١: الدالة/المختصر assert

أولا قمنا بضم الملف الرأسي assert.h في السطر الثاني، و استعملنا الدالة assert في كل من الـــسطر الخـــامس و السادس. في السطر الخامس قمنا بإعطاءه لدالة assert مقارنة صحيح، و في هذه الحالة سيتم تجاهل الدالة، و لن يقوم البرنامج بعمل أي شيء من طرف هذه الدالة. في السطر السادس أعطينا لدالة assert مقارنة خاطئة، و هنـــا ســـيتم التحويل إلى المختصرة assert و الذي سيقوم بطباعة الخطأ أثناء تشغيل البرنامج، و الخطأ هو *20<10* في الملف *main.c* في السطر ٦.

٣,٥,٢ الملف الرأسي ctype.h:

الملف الرأسي ctype.h يحتوي على مجموعة من الدوال الخاص أنواع الرموز، و اسمه مختصر من *character type*، و جميع دواله ذات وسيط واحد و هو لمتغير يحمل رقم الرمز.

٣,٥,٢,١ الدالة isalnum:

اسم الدالة مختصر من *is alphanumeric*، و تستعمل هذا الدالة لإختبار القيمة المعطاة لها، إذا كانت من القيمة لرمز حرفي أو رقمي فستكون النتيجة غير الصفر، أما في حالة العكس فنتيجة ستكون ٠، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5         int c = 1, c2 = 65;
6
7         printf("%d = %c\n", isalnum(c), c);
8         printf("%d = %c\n", isalnum(c2), c2);
9  }
```

البرنامج ٣,٥,٢: الدالة isalnum

في السطر السابع ستكون النتيجة ٠ أي خاطئة، لأن الرقم ١ في جدول *ASCII* عبارة عن أيقونة. في السطر الثامن ستكون النتيجة ١ أي صحيحة، لأن الرقم ٦٥ هو الحرف *A* في جدول *ASCII*، و هو من الرموز الحرفية.

٣,٥,٢,٢ الدالة isalpha:

اسم الدالة مختصر من *is alphabetic*، و تستعمل لإختبار القيمة المعطاة لها إذا كانت من حروف الأبجدية أم لا، إذا كان من الحروف الأبجدية فنتيجة ستكون ١، و في حالة العكس فستكون النتيجة ٠، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5         int c = 49, c2 = 65;
6
7         printf("%d = %c\n", isalpha(c), c);
8         printf("%d = %c\n", isalpha(c2), c2);
9  }
```

البرنامج ٣,٥,٣: الدالة isalpha

في السطر السابع، ستكون النتيجة ٠، لأن الرقم ٤٩ هو العدد ١ في جدول *ASCII*، و هو ليس من الحروف الأبجدية. في السطر الثامن ستكون النتيجة ١، لأن العدد ٦٥ هو الحرف *A* في جدول *ASCII*، و الحرف *A* من الحروف الأبجدية.

٣,٥,٢,٣ الدالة iscntrl:

اسم الدالة مختصر من *is control*، و هي تقوم بإختبار القيمة المعطاة لها إذا كانت من رموز التحكم، و رموز التحكم تكون بين ٠ و ٣١، و أيضا من ١٢٨ إلى ٢٥٥، و هنا ستكون النتيجة غير الصفر أي صحيحة، أما إذا كانت القيمة بين ٣٢ و ١٢٧ فستكون النتيجة خاطئة لأن كل من تلك الرموز عبارة عن أرقام و حروف لا أكثر، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5        int c = 65, c2 = 0;
6
7        printf("%d = %c\n", iscntrl(c), c);
8        printf("%d = %c\n", iscntrl(c2), c2);
9  }
```

البرنامج ٣,٥,٤: الدالة iscntrl

في السطر السابع النتيجة ستكون ٠ أي خاطئة، لأن الرقم ٦٥ يحمل الحرف *A* في جدول *ASCII* و هو ليس من رموز التحكم. في السطر الثامن ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم ٠ من رموز التحكم في جدول *ASCII*.

٣,٥,٢,٤ الدالة isdigit:

تقوم هذه الدالة بإختبار القيمة المعطاءة لها إذا كانت من الأرقام العشرية حيث تكون النتيجة غير الصفر إن كانت القيمة المعطاة بين الرقم ٤٨ و ٥٧، و في حالة أن النتيجة لم تكن بين الرقم ٤٨ و ٥٧ من جدول *ASCII* فستكون النتيجة ٠ أي خاطئة، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5        int c = 65, c2 = 48;
6
7        printf("%d = %c\n", isdigit(c), c);
8        printf("%d = %c\n", isdigit(c2), c2);
9  }
```

البرنامج ٣,٥,٥: الدالة isdigit

في السطر السابع ستكون النتيجة ٠ أي خاطئة، لأن الرقم ٦٥ هو الحرف *A* في جدول *ASCII* و الحرف *A* ليس رقما. في السطر الثامن ستكون النتيجة غير الصفر، لأن الرقم ٤٨ في جدول *ASCII* هو الرقم ٠.

٣,٥,٢,٥ الدالة isgraph:

تقوم هذا الدالة بإختبار القيمة المعطاة لها إذا كان رموز غير مرئية أو مرئية، إذا كانت غير مرئية فستكون النتيجة غير الصفر أي صحيحة، أما في حالة العكس فستكون النتيجة ٠ أي خاطئة، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
```

```
4  main(){
5        int c = 0, c2 = 48;
6
7        printf("%d = %c\n", isgraph(c), c);
8        printf("%d = %c\n", isgraph(c2), c2);
9  }
```

البرنامج ٣,٥,٦: الدالة isgraph

في السطر السابع ستكون النتيجة ٠ أي خاطئة، لأن الرقم ٠ في جدول *ASCII* عبارة عن رمز مرئي. في السطر الثامن ستكون النتيجة غير الصفر، لأن الرقم ٤٨ في جدول *ASCII* هو الرقم ٠ و هو رمز غير مرئي.

٣,٥,٢,٦ الدالة islower:

تقوم هذا الدالة بإختبار القيمة المعطاة لها، إذا كانت من الأحرف الصغيرة فستكون النتيجة غير الصفر أي صحيحة، أما إذا كانت غير ذلك فستكون النتيجة ٠، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5        int c = 65, c2 = 97;
6
7        printf("%d = %c\n", islower(c), c);
8        printf("%d = %c\n", islower(c2), c2);
9  }
```

البرنامج ٣,٥,٧: الدالة islower

في السطر السابع ستكون النتيجة ٠ أي خاطئة، لأن الرقم ٦٥ هو الرمز *A* في جدول *ASCII*، و الرمز *A* ليس من الحروف الصغيرة. في السطر الثامن ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم ٩٧ في جدول *ASCII* هو الرمز a و هو من الرمز الصغيرة.

٣,٥,٢,٧ الدالة isprint:

تقوم هذه الدالة بإختبار القيمة المعطاة لها، إذا لم كانت القيمة بين ٣٢ و ١٢٦ من جدول *ASCII* فستكون النتيجة غير الصفر، و هي الرموز المستعملة في الطباعة الإفتراضية (الفراغ يعتبر رمز أيضا)، أما إذا كانت القيم غير ذلك فستكون النتيجة ٠ أي خاطئة، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5        int c = 1, c2 = 65;
```

```
6
7       printf("%d = %c\n", isprint(c), c);
8       printf("%d = %c\n", isprint(c2), c2);
9 }
```

البرنامج ٣,٥,٨: الدالة isprint

في السطر السابع ستكون النتيجة ٠، لأن الرقم ١ في جدول *ASCII* عبارة عن رسم لإبتسامة، و لا تستعمل الإبتسامات في النصوص الإفتراضية. في السطر الثامن ستكون النتيجة غير الصفر، لأن الرقم ٦٥ هو الحرف *A* في جدول *ASCII* و هو حرف يمكن استعماله في النصوص الإفتراضية.

٣,٥,٢,٨ الدالة ispunct:

اسم الدالة مختصر من *is punctuation*، و هي تقوم بإختبار القيمة المعطاة لها، إذا كانت من أحد رموز الترقين فستكون النتيجة غير الصفر أي صحيحة، أما غير رموز الترقين فستكون النتيجة ٠ أي خاطئة، مثال:

```
1 #include<stdio.h>
2 #include<ctype.h>
3
4 main(){
5       int c = 44, c2 = 65;
6
7       printf("%d = %c\n", ispunct(c), c);
8       printf("%d = %c\n", ispunct(c2), c2);
9 }
```

البرنامج ٣,٥,٩: الدالة ispunct

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم ٤٤ هو الفاصلة في جدول *ASCII*، الفاصلة من رموز الترقين. في السطر الثامن ستكون النتيجة صفر أي خاطئة، لأن الرقم ٦٥ هو الحرف *A* في جدول *ASCII*، و الحروف ليست رموز ترقين.

٣,٥,٢,٩ الدالة isspace:

تقوم الدالة بإختبار القيمة المعطاة لها، إذا كانت من رموز الفضاء الأبيض أما لا، و رموز الفضاء الأبيض تبدأ من ٩ إلى ١٣، و في هذه الحالة ستكون النتيجة غير الصفر أي صحيح، أما إن لم تكون القيمة المعطاة من أحد الرموز الفضاء الأبيض فإن النتيجة ستكون ٠ أي خاطئة، مثال:

```
1 #include<stdio.h>
2 #include<ctype.h>
3
4 main(){
5       int c = 9, c2 = 97;
```

```
6
7        printf("%d = %c\n", isspace(c), c);
8        printf("%d = %c\n", isspace(c2), c2);
9  }
```

البرنامج ٣,٥,١٠: الدالة isspace

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم ٩ في جدول *ASCII* هو من أحد رموز الفضاء الأبيض. في السطر الثامن ستكون النتيجة ٠ أي خاطئة، لأن الرقم ٩٧ في جدول *ASCII* هو الحرف a، و هو ليس من رموز الفضاء الأبيض.

٣,٥,٢,١٠ الدالة isupper:

تقوم هذا الدالة بإختبار القيمة المعطاة لها، إذا كانت من الرموز الكبيرة فستكون النتيجة غير الصفر أي صحيحة، أما إذا كانت قيم لأحرف صغيرة فستكون النتيجة ٠ أي خاطئة، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5        int c = 65, c2 = 97;
6
7        printf("%d = %c\n", isupper(c), c);
8        printf("%d = %c\n", isupper(c2), c2);
9  }
```

البرنامج ٣,٥,١١: الدالة isupper

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم ٦٥ في جدول *ASCII* هو الحرف *A* و هو من الأحرف الكبيرة. في السطر الثامن ستكون النتيجة ٠ أي خاطئة، لأن الرقم ٩٧ في جدول *ASCII* هو الحرف *a* و هو ليس حرف كبير.

٣,٥,٢,١١ الدالة isxdigit:

اسم هذا الدالة مختصر من *is hexadecimal digital*، و تقوم هذا الدالة بإختبار الدالة المعطاة لها، إذا كانت القيمة من أعداد النظام السداسي عشر أي من *A-F* و من ٠ – ٩ فستكون النتيجة غير الصفر أي صحيحة، أما إذا كانت غير تلك القيم فستكون النتيجة ٠ أي خاطئة، مثال:

```
1  #include<stdio.h>
2  #include<ctype.h>
3
4  main(){
5        int c = 70, c2 = 71;
```

```
6
7         printf("%d = %c\n", isxdigit(c), c);
8         printf("%d = %c\n", isxdigit(c2), c2);
9   }
```

البرنامج ٣,٥,١٢: الدالة isxdigit

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم ٧٠ هو الحرف *F* في جدول *ASCII*، و الحرف *F* من النظام السداسي عشر و في النظام العشري هو العدد ١٥. في السطر الثامن ستكون النتيجة صفر أي خاطئة، لأن الرقم ٧١ هو الحرف *G* في جدول *ASCII*، و هو الحرف *G* ليس من النظام السداسي عشر.

٣,٥,٢,١٢ الدالتين toupper و tolower:

الدالة toupper تقوم بتحويل الأحرف الصغير إلى الأحرف الكبيرة، و الدالة tolower تقوم بتحويل الأحرف الكبيرة إلى أحرف صغيرة أي عكس ما تقوم به الدالة toupper، مثال:

```
1   #include<stdio.h>
2   #include<ctype.h>
3
4   main(){
5         int c = 'a', c2 = 'B';
6
7         printf("%c = %c\n", c, toupper(c));
8         printf("%c = %c\n", c2, tolower(c2));
9   }
```

البرنامج ٣,٥,١٣: الدالتين toupper و tolower

في هذا البرنامج يتم طباعة الحرف *a* و هو على شكل حرف صغير، ثم إعادة كتابته بعد تمريره لدالة toupper حيث سيصبح من الأحرف الكبيرة، و ستتم طباعة الحرف *B* على شكل حرف كبير، ثم إعادة كتابته بعد تمريره لدالة tolower حيث تحويله إلى حرف صغير.

٣,٥,٣ الملف الرأسي errno.h:

يوجد في الملف الرأسي errno.h مجموعة من المختصرات تستعمل لكشف الأخطاء أثناء وقت تشغيل البرنامج. حيث تستعمل تلك المختصرات مع الدالة perror.

٣,٥,٣,١ الدالة perror:

اسم الدالة perror مختصر من *print error*، و هي دالة تقوم بطباعة أخطاء خاصة بالملفات. لدالة وسيط واحد و هو سلسلة لحروف التي ستتم طباعتها قبل طباعة الخطأ، أو يمكن ترك السلسلة فراغه إن لم نريد إضافة شـيء إلى رسـالة الخطأ، مثال حول طريقة استعمال الدالة:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<errno.h>
4
5  main(){
6          FILE *File;
7          char FileName[255];
8
9          perror("situation");
10
11         printf("Enter The File Name: ");
12         gets(FileName);
13
14         File = fopen(FileName, "r");
15
16         if(!File){
17                 perror("situation");
18                 printf("errno = %d\n", errno);
19                 exit(1);
20         }else{
21                 perror("situation");
22                 printf("File Founded!\n");
23                 fclose(File);
24         }
25 }
```

البرنامج ٣,٥,١٤: الدالة perror

في السطر التاسع قمنا باستعمال الدالة perror مع تمرير لها النص *situation* و التي ستكون هذه الكلمة قبـل رسـالة الخطأ، و لكن هنا لا يوجد أي خطأ و الرسالة ستكون *No error*. استعمال الدالة perror في السطر السابع عشر، فإن كان اسم الملف الذي طلبه المستخدم غير موجود فسيتم طباعة الرسالة عدم وجود الملف، و أيـضا قمنـا باسـتعمال المختصر errno في السطر الثامن عشر عبر الدالة printf، حيث ستم طباعة رقم رسالة الخطأ الذي تم إيجاده، رسـالة الخطأ هنا رقمها ٢. و للمختصر errno ثوابت تستعمل لتبين على رسالة الخطأ التي نريد إرسالها، من أهم تلك الثوابت هي:

اسم الثابت	الرسالة
EINVAL	*Invalid argument*
ERANGE	*Result too large*
EBADF	*Bad file descriptor*
EDOM	*Domain error*
EACCES	*Permission denied*

Too many open files	EMFILE
File exists	EEXIST
Arg list too long	E2BIG
Improper link	EXDEV
No child processes	ECHILD
Resource temporarily unavailable	EAGAIN
Exec format error	ENOEXEC
Not enough space	ENOMEM
No such file or directory	ENOENT
No space left on device	ENOSPC

الجدول ٣,٥,١: ثوابت المختصر errno

و توجد رسائل أخرى يمكن رأيتها باستخدام التكرار، مثال:

```
1  #include<stdio.h>
2  #include<errno.h>
3  
4  main(){
5          int i;
6  
7          for(i=0;i<=42;i++){
8                  errno = i;
9                  printf("%d:", i);
10                 perror("");
11         }
12 }
```

البرنامج ٣,٥,١٥: الدالة perror (٢)

و كل رقم منها له ثابت معرف في الملف الرأسي errno.h، و هي على الشكل التالي:

```
1  #define EPERM           1
2  #define ENOENT          2
3  #define ESRCH           3
4  #define EINTR           4
5  #define EIO             5
6  #define ENXIO           6
7  #define E2BIG           7
8  #define ENOEXEC         8
9  #define EBADF           9
10 #define ECHILD          10
11 #define EAGAIN          11
12 #define ENOMEM          12
13 #define EACCES          13
14 #define EFAULT          14
15 #define EBUSY           16
16 #define EEXIST          17
17 #define EXDEV           18
18 #define ENODEV          19
19 #define ENOTDIR         20
20 #define EISDIR          21
21 #define EINVAL          22
22 #define ENFILE          23
```

```
23 #define EMFILE         24
24 #define ENOTTY         25
25 #define EFBIG          27
26 #define ENOSPC         28
27 #define ESPIPE         29
28 #define EROFS          30
29 #define EMLINK         31
30 #define EPIPE          32
31 #define EDOM           33
32 #define ERANGE         34
33 #define EDEADLK        36
34 #define ENAMETOOLONG   38
35 #define ENOLCK         39
36 #define ENOSYS         40
37 #define ENOTEMPTY      41
38 #define EILSEQ         42
```

البرنامج ٣,٥,١٦: ثوابت الملف الرأسي errno.h

٣,٥,٤ الملف الرأسي float.h:

يوجد بهذا الملف الرأسي مجموعة من الثابت خاصة بالأعداد الحقيقية، و تلك الثوابت هي:

اسم الثابت	قيمته
FLT_RADIX	2
FLT_ROUNDS	1
FLT_MAX	$1^{E}+37$
FLT_MAX_EXP	128
DBL_MAX	$1^{E}+37$
DBL_MAX_EXP	1024
FLT_DIG	6
DBL_DIG	15
FLT_EPSILON	$1^{E}-5$
DBL_EPSILON	$1^{E}-9$
FLT_MANT_DIG	24
DBL_MANT_DIG	53
FLT_MIN	$1^{E}-37$
FLT_MIN_EXP	-125
DBL_MIN	$1^{E}-37$
DBL_MIN_EXP	-1021

الجدول ٣,٥,٢: ثوابت الملف الرأسي float.h

تعتبر هذا الثوابت من أهم الثوابت الموجود في الملف الرأسي float.h، توجد ثوابت أخرى كثيرة يمكن رأيته في الملف الرأسي float.h.

٣,٥,٥ الملف الرأسي limits.h:

بهذا الملف الرأسي مجموعة من الثوابت لأحجام جميع الأنواع المتكاملة، الثوابت هي:

اسم الثابت	قيمته
CHAR_BIT	8
CHAR_MIN	-128
INT_MIN	-32767
LONG_MIN	-2147483647
SHRT_MIN	-32768
SCHAR_MIN	-128
LONG_MAX	2147483647
SCHAR_MAX	127
SHRT_MAX	32767
UCHAR_MAX	255
UINT_MAX	65535
ULONG_MAX	4294967295
USHRT_MAX	65535
INT_MAX	32767
CHAR_MAX	127

الجدول ٣,٥,١: ثوابت الملف الرأسي limits.h

و توجد ثوابت أخرى يمكن رأيتها في الملف الرأسي limits.h.

٣,٥,٦ الملف الرأسي locale.h:

لهذا الملف الرأسي مجموعة من تراكيب، ثوابت، مختصرات و دوال مستعمل من قبل روتينيات المواقــع (المنــاطق)، ثوابت هذا الملف الرأسي على الشكل التالي:

```
1  #define LC_ALL          0
2  #define LC_COLLATE      1
3  #define LC_CTYPE        2
4  #define LC_MONETARY     3
5  #define LC_NUMERIC      4
6  #define LC_TIME         5
7  #define LC_MIN          LC_ALL
8  #define LC_MAX          LC_TIME
```

البرنامج ٣,٥,١٧: ثوابت الملف الرأسي locale.h

لا يعتبر هذا الملف الرأسي مهم، هو خاص بتحديد المنطقة أو البلاد التي تريد استعمال أصنافها في برنامجك، منها اللغة. ألقي لنظرة على الملف الرأسي locale.h لرؤية باقي محتوياته.

٣,٥,٧ الملف الرأسي math.h:

يحمل هذا الملف الرأسي مجموعة من الثوابت و المختصرات و الدوال الخاص بالرياضيات. بالنسبة للثوابت فهي معرفة على الشكل التالي:

```
1  #define M_E         2.71828182845904523536
2  #define M_LOG2E     1.44269504088896340736
3  #define M_LOG10E    0.434294481903251827651
4  #define M_LN2       0.693147180559945309417
5  #define M_LN10      2.30258509299404568402
6  #define M_PI        3.14159265358979323846
7  #define M_PI_2      1.57079632679489661923
8  #define M_PI_4      0.785398163397448309616
9  #define M_1_PI      0.318309886183790671538
10 #define M_2_PI      0.636619772367581343076
11 #define M_2_SQRTPI  1.12837916709551257390
12 #define M_SQRT2     1.41421356237309504880
13 #define M_SQRT1_2   0.707106781186547524401
```

البرنامج ٣,٥,١٨: ثوابت الملف الرأسي math.h

أما المختصرات و الدوال فسندرس أهمها، أما الباقي فيمكن رأيتها من الملف الرأسي math.h.

٣,٥,٧,١ الدالة sin:

لدالة *sin* وسيط واحد و هو لمتغير من نوع double، و هي تقوم بإرجاع جيب *sine* القيمة التي تم تمريرها إليها، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5         double x = 3.14/2;
6
7         printf("sine x = %f\n", sin(x));
8  }
```

البرنامج ٣,٥,١٩: الدالة sin

٣,٥,٧,٢ الدالة cos:

اسم الدالة cos مختصر من *cosine*، و لها وسيط واحد من نوع double، و هي تقوم بإرجاع جيب التمام للقيمة التي تم تمريرها إليها، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5         double x = 3.14/2;
6
7         printf("cosine x = %f\n", cos(x));
8  }
```

البرنامج ٣,٥,٢٠: الدالة **cos**

٣,٥,٧,٣ الدالة tan:

اسم الدالة `tan` مختصر من *tangent*، و لها وسيط واحد لمتغير من نوع `double`،و هي تقوم بإرجاع الظلّ القيمة التي تم تمريرها إليها، مثال:

```
1 #include<stdio.h>
2 #include<math.h>
3
4 main(){
5       double x = 3.14/4;
6
7       printf("tangent x = %f\n", tan(x));
8 }
```

البرنامج ٣,٥,٢١: الدالة **tan**

٣,٥,٧,٤ الدالة exp:

اسم الدالة `exp` مختصر من *exponential*، و لها وسيط واحد من نوع `double`، حيث تكون القيمة المعطاة له هـي أسّ `e`، مثال:

```
1 #include<stdio.h>
2 #include<math.h>
3
4 main(){
5       double x = 3.0;
6
7       printf("exponential x = %f\n", exp(x));
8 }
```

البرنامج ٣,٥,٢٢: الدالة **exp**

٣,٥,٧,٥ الدالة log:

اسم الدالة `log` مختصر من *logarithm*، و بها وسيط واحد و هو لمتغير من نوع `double`، و تقوم الدالـة بإرجـاع لوغاريتمية القيمة التي تم تمريرها إليها، مثال:

```
1 #include<stdio.h>
2 #include<math.h>
3
4 main(){
5       double x = 100.0;
6
7       printf("logarithm x = %f\n", log(x));
8 }
```

البرنامج ٣,٥,٢٣: الدالة **log**

٣,٥,٧,٥ الدالة pow:

اسم الدالة pow مختصر من *power*، و لها وسيطين، الوسيط الأول هو متغير من نوع double، و الثاني أيضا متغير من نوع double، حيث الوسيط الثاني يكون عدد أسّ قيمة الوسيط الثاني، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=5.0, y=3.0;
6
7        printf("%f\n", pow(x, y));
8  }
```

البرنامج ٣,٥,٢٤: الدالة pow

٣,٥,٧,٦ الدالة sqrt:

اسم الدالة sqrt مختصر من *square*، و لها وسيط واحد لمتغير من نوع double، و هي تقوم بإرجاع الجذر التربيعي للقيمة المعطاة له، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=9.0;
6
7        printf("%f\n", sqrt(x));
8  }
```

البرنامج ٣,٥,٢٥: الدالة sqrt

٣,٥,٧,٧ الدالة ceil:

للدالة ceil وسيط واحد و هو لمتغير من نوع double، و هي تقوم بأخذ العدد المعطاة لها على شكل عدد صحيح، فإذا كانت القيمة المعطاة لها هي ٣,١ فستكون النتيجة ٤، أما إذا كانت ٣,٠ فستكون النتيجة هي نفسها ٣,٠، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=9.5;
6
7        printf("%f\n", ceil(x));
8  }
```

البرنامج ٣,٥,٢٦: الدالة ceil

٣,٥,٧,٨ الدالة floor:

الدالة floor هي معاكس لدالة ceil، فإذا كانت القيمة المعطاة لها ٣,١ فستكون النتيجة ٣,٠، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=10.5;
6
7        printf("%f\n", floor(x));
8  }
```

البرنامج ٣,٥,٢٧: الدالة floor

٣,٥,٧,٩ الدالة fabs:

اسم الدالة fabs مختصر من *float absolute*، و تقوم هذه الدالة بإرجاع القيمة المطلقة للعدد الذي تم تمريـــره لهـــا، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=-10.5, y=10.5;
6
7        printf("%f\n", fabs(x));
8        printf("%f\n", fabs(y));
9  }
```

البرنامج ٣,٥,٢٨: الدالة fabs

٣,٥,٧,١٠ الدالة ldexp:

لهذه الدالة وسيطين، الأول لمتغير من نوع double، و الوسيط الثاني متغير من نوع int، حيث تقوم هذه الدالـــة بضرب قيمة الوسيط الأول في ٢ أسّ قيمة الوسيط الثاني، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=2.0, y=4.0;
6
7        printf("%f\n", ldexp(x, y)); /*y = 2*2*2*2, x = 2; x*y = 32.000000*/
8  }
```

البرنامج ٣,٥,٢٩: الدالة ldexp

٣,٥,٧,١١ الدالة fmod:

لهذه الدالة وسيطين، كلاهما متغيران من نوع double، حيث تقوم هذه الدالة بقسمة قيمة الوسيط الأول على قيمة الوسيط الثاني، مثال:

```
1  #include<stdio.h>
2  #include<math.h>
3
4  main(){
5        double x=2.0, y=4.0;
6
7        printf("%f\n", fmod(x, y));
8  }
```

البرنامج ٣,٥,٣٠: الدالة fmod

٣,٥,٨ الملف الرأسي setjmp.h:

يحتوي هذا الملف الرأسي على دالتين هما setjmp و longjmp، حيث تحتوي الدالة setjmp على وسيط واحد و هو مؤشر لبنية jmp_buf و هي معرفة في نفس الملف الرأسي، و تحتوي الدالة longjmp على وسيطين، الأول للبنية jmp_buf و الوسيط الثاني لمتغير من نوع int. تستعمل الدالة setjmp مع الدالة longjmp، و الهدف منهما هو القفز إلى مختلف أماكن البرنامج، فمثلا نستعمل الكلمة المحجوزة goto للقفز من مكان لآخر في دالة معنية، و لا يمكن استعمالها للقفز العام (*Global*)، مثال:

```
1  #include<stdio.h>
2  #include<setjmp.h>
3  #include<stdlib.h>
4
5  void Return(void);
6  jmp_buf jmp;
7
8  main(){
9        int value;
10
11       value = setjmp(jmp);
12       {
13             printf("Block 1:\n");
14             if(value==1){
15                   printf("longjmp = %d\n", value);
16                   exit(1);
17             }
18       }
19
20       {
21             printf("Block 2:\n");
22             printf("Call Return...\n");
23             Return();
24       }
25 }
26
27 void Return(void){
```

```
28        longjmp(jmp, 1);
29  }
```

البرنامج ٣,٥,٣١: الدالة setjmp، و البنية jmp_buf

٣,٥,٩ الملف الرأسي signal.h:

يحتوي هذا الملف الرأسي على دالتين هما signal و raise، و الدالة المستعملة بكثرة هي الدالة raise.

٣,٥,٩,١ الدالة raise:

تحتوي هذه الدالة على وسيط من نوع int، و عملها هو إنهاء البرنامج في حالات مثل حدوث إنتهاك في ذاكرة الحاسوب، و توجد ثوابت خاصة بها في نفس الملف الرأسي، و كل ثابت و حالته، و هذا جدول لأهم الثوابت التي تستعمل مع هذه الدالة:

اسم الثابت	قيمته	الشرح
SIGABRT	٢٢	في حالة الإنتهاء الغير الطبيعي للبرنامج، يستدعي الدالة abort
SIGFPE	٨	في حالة خطأ رياضي
SIGILL	٤	في حالة حدوث أمر غير شرعي
SIGINT	٢	المقاطعة
SIGSEGV	١١	في حالة حدوث إنتهاك لذاكرة
SIGTERM	١٥	إنهاء البرنامج

الجدول ٣,٥,١: أهم الثوابت التي تستعمل مع الدالة raise

٣,٥,١٠ الملف الرأسي stdarg.h:

اسم هذا الملف الرأسي مختصر من *standard argument*، و هو خاص باستعمال الوسائط المتعددة لدوال. يحتوي الملف الرأسي stdarg.h على المؤشر va_list، و هو الذي سيحمل عدد الوسائط أو وسائط التي سيتم استعمالها في الدالة، و يجب دائما تشغيل *initialize* المؤشر باستخدام المختصر va_start، حيث يحتوي هذا المختصر على وسيطين، الوسيط الأول هو لمؤشر va_list الذي سيتم تشغيله، و الوسيط الثاني هو اسم الوسيط الأول لدالة. و بعد ذلك نستعمل المختصر va_arg، و الذي يحتوي على وسيطين، الوسيط الأول هو مؤشر لـ val_list الذي سيتم استعماله في الدالة، و الوسيط الثاني نضع فيه نوع الوسيط الذي سيتم أخذه، و في النهاية نقوم بإنهاء تشغيل المؤشر va_list و ذلك باستخدام المختصر va_end، مثال:

```
1  #include<stdio.h>
```

```
2  #include<stdarg.h>
3
4  void Arg_List(int list, ...); /*Function prototype*/
5
6  main(){
7        Arg_List(5, 10, 20, 30, 40, 50);
8  }
9
10 void Arg_List(int list, ...){
11       va_list pList;
12       int iArg=0, i;
13
14       printf("There are %d argument(s)\n", list);
15
16       va_start(pList, list); /*initialize the pList pointer*/
17
18       for(i=01;i<=list;i++){
19             iArg = va_arg(pList, int);
20             printf("argument %d = %d\n", i, iArg);
21       }
22
23       va_end(pList);
24
25 }
```

البرنامج ٣,٥,٣٢: الدالة `va_start`، الدالة `va_arg` و الدالة `va_end`، و المؤشر `va_list`

٣,٥,١١ الملف الرأسي `stddef.h`:

اسم هذا الملف الرأسي مختصر من *Standard Definition*، و هو يحتوي على الثابت NULL و هو معرف على الشكل التالي:

```
#define NULL      0
```

و يحتوي على المتغير size_t و هو معرف على الشكل التالي:

```
typedef unsigned size_t;
```

و طريقة استعماله كتالي:

```
1  #include<stdio.h>
2  #include<stddef.h>
3
4  main(){
5        size_t uInt = 65535;
6
7        printf("%u\n", uInt);
8  }
```

البرنامج ٣,٥,٣٣: المتغير size_t

و يحتوي أيضا على المتغير ptrdiff_t، و هو معرف بطريقتين هما:

```
typedef long      ptrdiff_t;
```

و الطريقة:

```
typedef int ptrdiff_t;
```

حيث يتم إختيار واحد من الطريتين حسب القيمة المعطاة أو المستعملة، مثال:

```
1  #include<stdio.h>
2  #include<stddef.h>
3
4  main(){
5        ptrdiff_t Int = 65535;
6        ptrdiff_t Long = 2147483647;
7
8        printf("Int = %d\n", Int);
9        printf("Long = %d\n", Long);
10 }
```

البرنامج ٣,٥,٣٤: المتغير ptrdiff_t

٣,٥,١٢ الملف الرأسي stdio.h:

يعبر من أهم الملفات الرأسية القياسية، و اسمه مختصر من *Standard Input Output*، حيث يحتوي على أهم الدوال و المختصرات التي يمكن استعمالها في أغلب البرامج، و دراسة جميع تلك الدوال و المختصرات بأمثل يأخذ الكثير من الصفحات لذا نكتفي بمعرفة أسماء تلك الدوال و المختصرات و الهدف منها. ينقسم هذا الملف الرأسي إلى أصناف، لكل صنف مجموعة من الدوال و المختصرات الخاصة به.

٣,٥,١٢,١ الدالة printf:

تحتوي هذه الدالة على وسائط غير محددة، تتزايد حسب رغبة المبرمج، و هي تقوم بالتعامل مع كل من الأحرف و الأرقام و النصوص، مثال:

```
1  #include<stdio.h>
2
3  main(){
```

```
4        int Number = 100;
5        char Char = 'A';
6        char String[] = "Hello, World!\n";
7
8        printf("%d\n", Number);
9        printf("%c\n", Char);
10       printf(String);
11 }
```

البرنامج ٣,٥,٣٥: الدالة printf

٣,٥,١٢,٢ الدالة sprintf:

أيضا هذه الدالة تحتوي على وسائط غير محدودة، و لكن لديه وسيط إضافي في بدايتها حيث هو عبارة عـــن مـــتغير لسلسلة حروف، و تقوم هذه الدالة بكتابة نص في تلك السلسلة النصية، مثال:

```
1  #include<stdio.h>
2
3  main(){
4        char Buffer[255];
5
6        sprintf(Buffer, "Number = %d\nCharacter = %c\nString = %s\n",
7              100, 'A', "Hello, World!");
8
9        printf(Buffer);
10 }
```

البرنامج ٣,٥,٣٦: الدالة sprintf

٣,٥,١٢,٣ الدالة vprintf:

مثل الدالة printf و لكنها تأخذ الوسائط على شكل va_list، مثال:

```
1  #include<stdio.h>
2  #include<stdarg.h>
3
4  void ArgLst(char *format, ...);
5
6  main(){
7        int Number = 100;
8        char Char = 'A';
9        char String[] = "Hello, World!";
10
11       ArgLst("Number = %d\nChar = %c\nStrin = %s\n",\
12             Number, Char, String);
13 }
14
15 void ArgLst(char *format, ...){
16       va_list Lst;
17
18       va_start(Lst, format);
19       vprintf(format, Lst);
20       va_end(Lst);
21 }
```

البرنامج ٣,٥,٣٧: الدالة **vprintf**

٣,٥,١٢,٤ الدالة vfprintf:

هي مطابقة لدالة vprintf فقط هي خاصة بالتعامل مع الملفات، مثال:

```
1  #include<stdio.h>
2  #include<stdarg.h>
3
4  void ArgLst(char *format, ...);
5
6  main(){
7        int Number = 100;
8        char Char = 'A';
9        char String[] = "Hello, World!";
10
11       ArgLst("Number = %d\nChar = %c\nStrin = %s\n",\
12             Number, Char, String);
13 }
14
15 void ArgLst(char *format, ...){
16       va_list Lst;
17       FILE *File;
18
19       File = fopen("Info.txt", "w");
20
21       va_start(Lst, format);
22       vfprintf(File, format, Lst);
23       va_end(Lst);
24       fclose(File);
25 }
```

البرنامج ٣,٥,٣٨: الدالة **vfprintf**

٣,٥,١٢,٥ الدالة vsprintf:

هذه الدالة مدمجة مع كل من الدالة sprintf و الدالة vprintf، مثال:

```
1  #include<stdio.h>
2  #include<stdarg.h>
3
4  void ArgLst(char *format, ...);
5
6  main(){
7        int Number = 100;
8        char Char = 'A';
9        char String[] = "Hello, World!";
10       char buffer[255];
11
12       ArgLst(buffer, "Number = %d\nChar = %c\nStrin = %s\n",\
13             Number, Char, String);
14
15       printf(buffer);
16 }
```

```
17
18 void ArgLst(char *buffer, char *format, ...){
19 	va_list Lst;
20 
21 	va_start(Lst, format);
22 	vsprintf(buffer, format, Lst);
23 	va_end(Lst);
24 }
```

البرنامج ٣,٥,٣٩: الدالة vsprintf

٣,٥,١٢,٦ الدالة scanf:

تقوم هذه الدالة باستقبال الرموز من لوحة المفاتيح، و هي دالة خاصة بالإدخال، مثال:

```
1 #include<stdio.h>
2 
3 main(){
4 	char string[255];
5 
6 	printf("Your name, Please: ");
7 	scanf("%s", string);
8 	printf("Nice to meet you %s!\n", string);
9 }
```

البرنامج ٣,٥,٤٠: الدالة scanf

٣,٥,١٢,٧ الدالة fscanf:

تقوم هذه الدالة بأخذ رموز من ملف نصي إما على شكل أرقام أو أحرف أو سلسلة نصوص، مثال:

```
1 #include<stdio.h>
2 
3 main(){
4 	FILE *File;
5 	char buffer[255];
6 
7 	File = fopen("Info.txt", "r");
8 
9 	fscanf(File, "%s", buffer);
10 
11 	printf("%s\n", buffer);
12 }
```

البرنامج ٣,٥,٤١: الدالة fscanf

٣,٥,١٢,٨ الدالة sscanf:

تقوم هذه الدالة بنشر سلسلة حروف على مجموعة من المتغيرات بشكل منفصل، مثال:

```
1 #include<stdio.h>
2 
3 main(){
4 	char buffer[] = "Hello, World! 15";
```

```
5        char string1[255], string2[255];
6        int number;
7
8        sscanf(buffer, "%s%s%d", string1, string2, &number);
9
10
11       printf("String = %s %s\n", string1, string2);
12       printf("Number = %d\n", number);
13  }
```

البرنامج ٣,٥,٤٢: الدالة **sscanf**

٣,٥,١٢,٩ الدالة **fgetc**:

تقوم هذه الدالة بأخذ أحرف من ملف نصي، حيث كلما يتم استعمال هذه الدالة يتقدم مؤشر الملف بخطوة، مثال:

```
1   #include<stdio.h>
2
3   main(){
4        FILE *File;
5        char ch;
6
7        File = fopen("Info.txt", "r");
8
9        while(feof(File)==0){
10             ch = fgetc(File);
11             printf("%c", ch);
12       }
13  }
```

البرنامج ٣,٥,٤٣: الدالة **fgetc**

٣,٥,١٢,١٠ الدالة **fgets**:

مثل الدالة fgetc فقط تأخذ سطر كامل بدل حرف واحد، مثال:

```
1   #include<stdio.h>
2
3   main(){
4        FILE *File;
5        char buffer[255];
6
7        File = fopen("Info.txt", "r");
8
9        while(feof(File)==0){
10             fgets(buffer, 255, File);
11             printf(buffer);
12       }
13
14       fclose(File);
15  }
```

البرنامج ٣,٥,٤٤: الدالة **fgets**

٣,٥,١٢,١١ الدالة fputc:

تقوم هذه الدالة بكتابة حرف إلى ملف نصي، مثال:

```
1  #include<stdio.h>
2
3  main(){
4         FILE *File;
5
6         File = fopen("Info.txt", "w");
7
8         fputc('A', File);
9         fclose(File);
10 }
```

البرنامج ٣,٥,٤٥: الدالة fputc

٣,٥,١٢,١٢ الدالة fputs:

تقوم هذه الدالة بكتابة سلسلة حرفية في ملف نصي، مثال:

```
1  #include<stdio.h>
2
3  main(){
4         FILE *File;
5
6         File = fopen("Info.txt", "w");
7
8         fputs("Hello, World!", File);
9         fclose(File);
10 }
```

البرنامج ٣,٥,٤٩: الدالة fputs

٣,٥,١٢,١٣ الدالة getc:

مثل الدالة fgetc و لكنها لا تحتوي على وسائط، مثال:

```
1  #include<stdio.h>
2
3  main(){
4         FILE *File;
5         char  ch;
6
7         File = fopen("Info.txt", "r");
8
9         while(feof(File)==0){
10               ch = getc(File);
11               printf("%c", ch);
12        }
13
14        fclose(File);
```

```
15 | }
```

البرنامج ٣,٥,٤٧: الدالة getc

٣,٥,١٢,١٤ الدالة getchar:

تقوم هذه الدالة بالإدخال، حيث تستقبل حرف واحد فقط من المستخدم، مثال:

```
1 #include<stdio.h>
2 
3 main(){
4 	char ch;
5 
6 	printf("Enter a character: ");
7 	ch = getchar();
8 	printf("Your character is: %c\n", ch);
9 }
```

البرنامج ٣,٥,٤٨: الدالة getchar

٣,٥,١٢,١٥ الدالة gets:

تقوم هذه الدالة الإدخال أيضا، و هي تستقبل سلسلة حرفية من المستخدم، مثال:

```
1 #include<stdio.h>
2 
3 main(){
4 	char str[255];
5 
6 	printf("Enter a string(Max character 255): ");
7 	gets(str);
8 	printf("Your string are: %s\n", str);
9 }
```

البرنامج ٣,٥,٤٩: الدالة gets

٣,٥,١٢,١٦ الدالة putc:

مثل الدالة fputc، مثال:

```
1 #include<stdio.h>
2 
3 main(){
4 	FILE *File;
5 
6 	File = fopen("Info.txt", "w");
7 
8 	putc('A', File);
9 	fclose(File);
10 }
```

البرنامج ٣,٥,٥٠: الدالة putc

٣,٥,١٢,١٧ الدالة putchar:

تقوم هذه الدالة بطباعة حرف على شاشة الحاسوب، مثال:

```
1  #include<stdio.h>
2
3  main(){
4        char ch = 'A';
5
6        printf("ch = ");
7        putchar(ch);
8        putchar('\n');
9  }
```

البرنامج ٣,٥,٥١: الدالة putchar

٣,٥,١٢,١٨ الدالة puts:

تقوم هذه الدالة بطباعة سلسلة حرفية على شاشة الحاسوب،حيث يتم الرجوع إلى سطر جديـد في كـل سلـسلة حرفية، مثال:

```
1  #include<stdio.h>
2
3  main(){
4        char str1[] = "Line 1: Hello, World!";
5        char str2[] = "Line 2: Hello, World!";
6
7        puts(str1);
8        puts(str2);
9  }
```

البرنامج ٣,٥,٥٢: الدالة puts

٣,٥,١٢,١٩ الدالة ungetc:

تقوم هذه الدالة بحذف حرف من ملف نصي، مثال:

```
1  #include<stdio.h>
2
3  main(){
4        FILE *File;
5
6        File = fopen("Info.txt", "w");
7
8        ungetc('A', File);
9        fclose(File);
10 }
```

البرنامج ٣,٥,٥٣: الدالة ungetc

٣,٥,١٢,٢٠ الدالة fopen:

تستعمل هذه الدالة لقراءة و كتابة الملفات، حيث تحتوي على وسيطين، الوسيط الأول لسلسلة حرفية التي هي اسم الملف الذي سيتم التعامل معه، و الوسيط الثاني هو النمط الذي سيتم استعماله مع الملف، حيث يكون ذلك النمط إمـــا للكتابة أو القراءة أو كلهما على الملفات، مثال:

```
1  #include<stdio.h>
2  
3  main(){
4  	FILE *File;
5  
6  	File = fopen("FileName.Ext", "r");
7  
8  	if(File==NULL)
9  		printf("File does not exist!\n");
10 	else
11 		printf("File exist now!\n");
12 
13 	File = fopen("FileName.Ext", "w");
14 	if(File!=NULL)
15 		printf("File Created!\n");
16 
17 	if(File==NULL)
18 		printf("File does not exist!\n");
19 	else
20 		printf("File exist now!\n");
21 
22 	fclose(File);
23 }
```

البرنامج ٣,٥,٥٤: الدالة fopen

٣,٥,١٢,٢١ الدالة freopen:

تستعمل هذه الدالة مثل الدالة fopen مع وسيط إضافي لمؤشر بينة FILE و الذي يكون إمـــا stdin، stdout أو stderr، مثال:

```
1  #include<stdio.h>
2  
3  main(){
4  	FILE *File;
5  
6  	File = freopen("Info.txt", "w", stderr);
7  
8  	if(!File)
9  		fprintf(stdout, "Error!\n");
10 	else
11 		fprintf(File, "String!");
12 
13 	fclose(File);
14 }
```

البرنامج ٣,٥,٥٥: الدالة freopen

٣,٥,١٢,٢٢ الدالة fclose:

تقوم هذه الدالة بغلق ملف حيث يمكن استعماله مرة أخرى، و توجد الدالة _fcloseall و هي تقوم بغلق جميـــع الملفات الغير مغلق و ترجع قيمة لعدد الملفات التي تم إغلاقها، مثال:

```
1   #include<stdio.h>
2
3   main(){
4           FILE *File1, *File2, *File3;
5
6           File1 = fopen("Info1.txt", "w");
7           File2 = fopen("Info2.txt", "w");
8           File3 = fopen("Info3.txt", "w");
9
10          fclose(File1);
11
12          printf("%d file(s) closed by _fcloseall()\n",\
13                  _fcloseall());
14  }
```

البرنامج ٣,٥,٥٦: الدالة fclose

٣,٥,١٢,٢٣ الدالة remove:

تقوم هذه الدالة بحذف الملفات، مثال:

```
1   #include<stdio.h>
2
3   main(){
4           FILE *File;
5
6           /*Create temporary.tmp*/
7           File = fopen("temporary.tmp", "w");
8           fclose(File);
9
10          /*remove temporary.tmp*/
11          remove("temporary.tmp");
12  }
```

البرنامج ٣,٥,٥٧: الدالة remove

٣,٥,١٢,٢٤ الدالة rename:

تقوم هذه الدالة بإعادة تسمية ملف، مثال:

```
1   #include<stdio.h>
2
3   main(){
4           printf("Rename Info.txt To Data.dat...\n");
5           rename("Info.txt", "Data.dat");
6           printf("Operation Terminate...\n");
```

```
7 | }
```

البرنامج ٣,٥,٥٨: الدالة rename

٣,٥,١٢,٢٥ الدالة tmpfile:

تقوم هذه الدالة بإنشاء ملف مؤقت *temporary file* للبرنامج، حيث يتم حذف الملف المؤقت عند الخروج من البرنامج أو نهاية البرنامج، مثال:

```
1  #include<stdio.h>
2
3  main(){
4          FILE *File;
5
6          printf("Create a Temporary File...\n");
7          File = tmpfile();
8          printf("Temporary File Created...\n");
9
10         /*At exit*/
11         printf("Temporary File deleted...\n");
12 }
```

البرنامج ٣,٥,٥٩: الدالة tmpfile

٣,٥,١٢,٢٦ الدالة fread:

تقوم هذه الدالة بقراءة محتوى ملف و وضع نسخة لها في سلسلة حرفية، و تحتوي على أربعة وسائط، الوسيط الأول هو لسلسلة الحرفية التي سيتم وضع فيها نسخة من نص الملف، و الوسيط الثاني هو حجم المحتوى الواحد بالبايتات، و الوسيط الثالث هو عدد الأحرف التي نريد نسخها، و الوسيط الأخير و مؤشر للبنية FILE، مثال:

```
1  #include<stdio.h>
2
3  main(){
4          FILE *File;
5          char buffer[255];
6
7          File = fopen("Info.txt", "r");
8
9          fread(buffer, sizeof(char), 255, File);
10         printf("%s\n", buffer);
11 }
```

البرنامج ٣,٥,٦٠: الدالة fread

٣,٥,١٢,٢٧ الدالة fwrite:

تقوم هذه الدالة بالكتابة على الملفات، و هي الدالة المعاكسة لدالة fread، و لها نفس وسائط الدالة fread، مثال:

```
1  #include<stdio.h>
2
```

```
3  main(){
4        FILE *File;
5        char buffer[] = "String!";
6
7        File = fopen("Info.txt", "w");
8
9        fwrite(buffer, sizeof(char), 7, File);
10       fclose(File);
11 }
```

البرنامج ٣,٥,٦١: الدالة **fwrite**

٣,٥,١٢,٢٨ الدالة fseek:

تقوم هذه الدالة بالتحكم في مكان مؤشر لملف نصي، حيث تحتوي على ثلاثة وسائط، الوسيط الأول هـو مؤشـر للبنية FILE و هو الملف الذي سيتم تحديد مكان مؤشره، و الوسيط الثاني هو متغير من نوع long و هو عدد البايتات من الوسيط الثالث، مثلا إذا أعطيناه القيمة ٥ فسيتم تجاهل خمسة بايتات بعد المؤشر الذي تم تحديده في الوسيط الثالث، و أخيرا الوسيط الثالث، و هو متغير من نوع int، و من هذا الوسيط نقوم بتحديد المؤشر لنص، و له ثوابت و هـي SEEK_CUR و SEEK_SET و SEEK_END، مثال:

```
1  #include<stdio.h>
2
3  main(){
4        FILE *File;
5        char ch_set, ch_cur, ch_end;
6
7        File = fopen("Info.txt", "r");
8
9        /*Beginning of file*/
10       fseek(File, 0L, SEEK_SET);
11
12       printf("SEEK_SET Begin:\n");
13       while(feof(File)==0){
14             ch_set = fgetc(File);
15             printf("%c", ch_set);
16       }
17       printf("\nSEEK_SET End.\n");
18
19       /*Current position of file pointer*/
20       fseek(File, 0L, SEEK_CUR);
21
22       printf("SEEK_CUR Begin:\n");
23       while(feof(File)==0){
24             ch_cur = fgetc(File);
25             printf("%c", ch_cur);
26       }
27       printf("\nSEEK_CUR End.\n");
28
29       /*End of file*/
30       fseek(File, 0L, SEEK_END);
31
32       printf("SEEK_END Begin:\n");
```

```
33        while(feof(File)==0){
34              ch_end = fgetc(File);
35              printf("%c", ch_end);
36        }
37        printf("\nSEEK_END End.\n");
38  }
```

البرنامج ٣,٥,٦٢: الدالة fseek

٣,٥,١٢,٢٩ الدالة ftell:

تقوم هذه الدالة بإرجاع قيمة لمكان مؤشر النص الحالي، مثال:

```
1   #include<stdio.h>
2
3   main(){
4         FILE *File;
5         int Position;
6         char buff[3];
7
8         File = fopen("Info.txt", "r");
9
10        /* Move the pointer of file 3 bytes
11        by reading character*/
12        fread(buff, sizeof(char), 3, File);
13
14        Position = ftell(File);
15        printf("Position after read 3 characters is %d\n",\
16              Position);
17  }
```

البرنامج ٣,٥,٦٣: الدالة ftell

٣,٥,١٢,٣٠ الدالة rewind:

تقوم هذه الدالة بإرجاع مؤشر ملف إلى البداية، و هي مكافئة لــ fseek(File, 0L, SEEK_SET)، مثال:

```
1   #include<stdio.h>
2
3   main(){
4         FILE *File;
5         char ch;
6
7         File = fopen("Info.txt", "r");
8
9         fseek(File, 0L, SEEK_END);
10        printf("fseek(File, 0L, SEEK_END):\n");
11        while(feof(File)==0){
12              ch = fgetc(File);
13              printf("%c", ch);
14        }
15
16        printf("\n----------------\n");
17
18        rewind(File);
```

```
19        printf("rewind(File):\n");
20        while(feof(File)==0){
21              ch = fgetc(File);
22              printf("%c", ch);
23        }
24
25        printf("\n");
26  }
```

البرنامج ٣,٥,٦٤: الدالة rewind

٣,٥,١٢,٣١ الدالة feof:

تقوم هذه الدالة بإختبار إذا كان مؤشر ملف قد وصل إلى نهاية الملف، حيث ترجع هذه الدالة القيمة ٠ إذا مؤشر الملف في النهاية، و ترجع قيمة غير الصفر إن لم يكن المؤشر في نهاية الملف، مثال:

```
1   #include<stdio.h>
2
3   main(){
4         FILE *File;
5
6         File = fopen("Info.txt", "r");
7
8         fseek(File, 0L, SEEK_END);
9         if(feof(File)==0)
10              printf("Position of pointer is in the end of file\n");
11        else
12              printf("Position of pointer is not in the end of file\n");
13  }
```

البرنامج ٣,٥,٦٥: الدالة feof

٣,٥,١٣ الملف الرأسي stdlib.h:

الاسم *stdlib* مختصر من *Standard Library*، حيث يحتوي هذا الملف الرأسي على مجموعة من الدوال في كل من دوال تحويل الأرقام، دوال تخصيص الذاكرة (التخزين) و دوال أخرى.

٣,٥,١٣,١ الدالة atof:

تقوم هذه الدالة بتحويل أعداد موجود في سلسلة حرفية إلى أعداد حقيقية من نوع double، حيث يمكن التعامل مع تلك الأرقام بسهولة، مثال:

```
1   #include<stdio.h>
2   #include<stdlib.h>
3
4   main(){
5         char str[3] = "123";
6         double num;
7
```

```
8          num = atof(str);
9
10         printf("%f\n", num);
11  }
```

البرنامج ٣,٥,٦٦: الدالة atof

٣,٥,١٣,٢ الدالة atoi:

تقوم هذه الدالة بنفس ما تقوم به الدالة atof إلا أنها تحول أعداد السلسلة الحرفية إلى أعداد صحيحة من نوع int، مثال:

```
1   #include<stdio.h>
2   #include<stdlib.h>
3
4   main(){
5          char str[3] = "123";
6          int num;
7
8          num = atoi(str);
9
10         printf("%d\n", num);
11  }
```

البرنامج ٣,٥,٦٧: الدالة atoi

٣,٥,١٣,٣ الدالة atol:

تقوم هذه الدالة بنفس ما تقوم به الدالة atoi إلا أنها تحول أعداد السلسلة الحرفية إلى أعداد صحيحة من نوع long، مثال:

```
1   #include<stdio.h>
2   #include<stdlib.h>
3
4   main(){
5          char str[3] = "123";
6          long num;
7
8          num = atoi(str);
9
10         printf("%ld\n", num);
11  }
```

البرنامج ٣,٥,٦٨: الدالة atol

٣,٥,١٣,٤ الدالة rand:

تقوم هذه الدالة بتوليد أرقام عشوائية، مثال:

```
1   #include<stdio.h>
2   #include<stdlib.h>
```

```
3
4   main(){
5         int i, j=0;
6         for(i=0;i<=10;i++){
7               j=rand()%100;
8               printf("j = %d\n", j);
9         }
10  }
```

البرنامج ٣,٥,٦٩: الدالة rand

و هنا لن تتعد الأعداد العشوائية العدد ١٠٠، و يمكننا أن نجعلها أكثر أو أقل من ذلك فقط نقوم بالتغير في العدد ١٠٠ الذي موجود بعد رامز باقي القسمة.

٣,٥,١٣,٥ الدالة srand:

تقوم هذه الدالة بتشغيل مولد الأرقام العشوائية rand، حيث يجعل تلك القيم العشوائية متغير من لحظة لأخرى عندما نمرر له الدالة time، مثال:

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<time.h>
4
5   main(){
6         int i;
7
8         srand(time(NULL));
9
10        for(i=0;i<=10;i++){
11              printf("%d\n", rand()%100);
12        }
13  }
```

البرنامج ٣,٥,٧٠: الدالة srand

٣,٥,١٣,٦ الدالة abort:

تستعمل هذه الدالة في حالة وجود أخطاء في البرنامج، حيث تقوم بإيقاف البرنامج و إظهار رسالة تخبر المستخدم أن طريقة إنتهاء البرنامج غير طبيعية، مثال:

```
1   #include<stdio.h>
2   #include<stdlib.h>
3
4   main(){
5         FILE *File;
6
7         if(!(File=fopen("FileName.txt", "r"))){
8               printf("Can't open file\n");
9               abort();
```

```
10        }
11
12        fclose(File);
13  }
```

البرنامج ٣,٥,٧١: الدالة **abort**

٣,٥,١٣,٧ الدالة exit:

تقوم هذه الدالة بالإنهاء البرنامج إذا تم التمرير إليها القيمة ١، حيث يوجد ثوابت باسم EXIT_FAILURE و هو نفسه القيمة ١، و الثابت EXIT_SUCCESS و هو القيمة ٠ و الذي يعني الخروج السليم للبرنامج، مثال:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  main(){
5        FILE *File;
6
7        if(!(File=fopen("FileName.txt", "r"))){
8              printf("Can't open file\n");
9              exit(EXIT_FAILURE);
10       }else
11             exit(EXIT_SUCCESS);
12
13       fclose(File);
14 }
```

البرنامج ٣,٥,٧٢: الدالة **exit**

٣,٥,١٣,٨ الدالة atexit:

تقوم هذه الدالة باستعداء دوال أخرى، حيث يتم تنفيذ تلك الدوال عند نهاية البرنامج، مثال:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  void AtExit(){
5        printf("Program - end\n");
6  }
7
8  main(){
9        atexit(AtExit);
10
11       printf("Program - start\n");
12 }
```

البرنامج ٣,٥,٧٣: الدالة **atexit**

٣,٥,١٣,٩ الدالة system:

تقوم هذه الدالة بتنفيذ أوامر النظام مثلا إذا أردنا مسح شاشة الجهاز باستخدام أوامر النظام نكتب:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  main(){
5
6        printf("Message!\n");
7
8        /*In DOS OS*/
9        system("cls");
10
11       /*In Unix/Linux OS*/
12       system("clear");
13 }
```

البرنامج ٣,٥,٧٤: الدالة **system**

٣,٥,١٣,١٠ الدالة abs:

تقوم هذه الدالة بإرجاع القيمة المطلقة من النوع int للعدد الذي تم تمريره إليها، مثال:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  main(){
5        int i=-10, j=20;
6
7        printf("absolute value if 'i' is %d\n", abs(i));
8        printf("absolute value if 'j' is %d\n", abs(j));
9  }
```

البرنامج ٣,٥,٧٥: الدالة **abs**

٣,٥,١٣,١١ الدالة labs:

نفس الدالة abs إلا أنها ترجع القيمة المطلق من النوع long، مثال:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  main(){
5        long i=-10, j=20;
6
7        printf("absolute value if 'i' is %d\n", labs(i));
8        printf("absolute value if 'j' is %d\n", labs(j));
9  }
```

البرنامج ٣,٥,٧٦: الدالة **labs**

٣,٥,١٣,١٢ الدالة div:

تقوم هذه الدالة بتقسيم قيمة الوسيط الأول على قيمة الوسيط الثاني، و الناتج يكون من نوع int، مثال:

```
1  #include<stdio.h>
2  #include<stdlib.h>
```

```
3
4  main(){
5        int a=12, b=6;
6
7        printf("%d/%d = %d\n", a, b, div(a, b));
8  }
```

البرنامج ٣,٥,٧٧: الدالة div

٣,٥,١٣,١٣ الدالة ldiv:

تقوم هذه الدالة بتقسيم قيمة الوسيط الأول على قيمة الوسيط الثاني، و الناتج يكون من نوع long، مثال:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  main(){
5        long a=12, b=6;
6
7        printf("%d/%d = %d\n", a, b, ldiv(a, b));
8  }
```

البرنامج ٣,٥,٧٨: الدالة ldiv

٣,٥,١٤ الملف الرأسي string.h:

في الملف الرأسي string.h توجد مجموعتين من الدوال، المجموعة الأولى تبدأ أسماءها بـــ str و هي تعني *string*، و المجموعة الثانية تبدأ أسماءها بـــ mem و هي تعني *memory*. سندرس أهم دوال هذا الملف الرأسي، و التي أغلبها تبـــدأ بـــ str.

٣,٥,١٤,١ الدالة strcpy و الدالة strncpy:

تقوم كلا من الدالتين بنسخ نص إلى سلسلة حرفية، الدالة الأولى strcpy بها وسيطين، الوسيط الأول هو سلـــسلة الحروف التي سيتم نسخ فيها النص، و الوسيط الثاني هو النص الذي سيتم نسخه. و الدالة الثانية strncpy بها ثلاثـــة وسائط، الوسيط الأول و الوسيط الثاني هما نفس الوسيط الأول و الثاني لدالة strcpy، أما الوسيط الثالث فهو عـــدد الأحرف التي نريد نسخها، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str[] = "Hello";
6        char empty[5];
7        char empty2[5];
8
9        strcpy(empty, str);
10       strncpy(empty2, str, 3);
11
```

```
12        printf("empty  = %s\n", empty);
13        printf("empty2 = %s\n", empty2);
14 }
```

البرنامج ٣,٥,٧٩: الدالة strcpy و الدالة strncpy

أما إذا كانت السلسلة empty بها نص سابقا، فسيتم حذفه، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str[] = "Hello";
6        char empty[5] = "empty";
7
8        strcpy(empty, str);
9
10       printf("empty  = %s\n", empty);
11 }
```

البرنامج ٣,٥,٨٠: الدالة strcpy (٢)

٣,٥,١٤,٢ الدالة strcat و الدالة strncat:

تقوم الدالة strcat بنسخ نص و إضافته إلى سلسلة حرفية، و أيضا الدالة strncat تقوم بنسخ نص و إضافته إلى سلسلة حرفية مع تحديد عدد الأحرف التي نريد إضافتها، مثال لطريقة استعمال الدالة strcat:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str[] = "Hello";
6
7        printf("str = %s\n", str);
8        strcat(str, ", World");
9        printf("str = %s\n", str);
10 }
```

البرنامج ٣,٥,٨١: الدالة strcat

و هذا مثال لطريقة استعمال الدالة strncat:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str[] = "Hello";
6
7        printf("str = %s\n", str);
8        strncat(str, ", World", 3);
9        printf("str = %s\n", str);
10 }
```

البرنامج ٣,٥,٨٢: الدالة strncat

٣,٥,١٤,٣ الدالة strcmp و الدالة strncmp:

تقوم الدالة strcmp بالمقارنة بين سلسلتين، إذا كانت السلسلة الحرفية الأولى هي الأكبر فستكون النتيجة أكبر من ٠، و في حالة أن السلسلة الحرفية الثانية هي الأكبر فستكون النتيجة أصغر من ٠، و أيضا الدالة strncmp تقوم بالمقارنة بين سلسلتين مع تحديد عدد حروف السلسلة الحرفية الأولى التي نريد مقارنتها مع السلسلة الحرفية الثانية، مثال حول طريقة استعمال الدالة strcmp:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str1[] = "Hello";
6        char str2[] = "Hello2";
7        int cmp = strcmp(str1, str2);
8
9        if(cmp>0)
10             printf("str1 > str2\n");
11       else
12             printf("str1 < str2\n");
13 }
```

البرنامج ٣,٥,٨٣: الدالة strcmp

و هذا مثال لطريقة استعمال الدالة strncmp:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str1[] = "Hello";
6        char str2[] = "Hello2";
7        int cmp = strncmp(str1, str2, 3);
8
9        if(cmp>0)
10             printf("str1 > str2\n");
11       else
12             printf("str1 < str2\n");
13 }
```

البرنامج ٣,٥,٨٤: الدالة strncmp

٣,٥,١٤,٤ الدالة strchr و الدالة strrchr:

تستعمل الدالة strchr للبحث عن مكان حرف في سلسلة حرفية، و إذا كان الحرف الذي نريد البحث عنه في السلسلة الحرفية موجود مرتين فسيتم أخذ مكان أول حرف، و الدالة strrchr مثل الدالة strchr و لكنها تأخذ الحرف الأخير بدل الأول، مثال:

```
1  #include<stdio.h>
```

```
2  #include<string.h>
3
4  main(){
5        char str[] = "Hello", *strdest;
6        char ch = 'l';
7        int result;
8
9        printf("%s\n", str);
10       printf("12345\n");
11
12       strdest = strchr(str, ch);
13       result = (int)(strdest-str+1);
14       printf("First '%c' in position %d\n", ch, result);
15
16       strdest = strrchr(str, ch);
17       result = (int)(strdest-str+1);
18       printf("Last '%c' in position %d\n", ch, result);
19 }
```

البرنامج ٣,٥,٨٥: الدالة strchr و الدالة strrchr

٣,٥,١٤,٥ الدالة strspn و الدالة strcspn:

باستخدام الدالة strspn يمكن معرفة عدد مجموعة من الأحرف في سلسلة حرفية، حيث يجب أن تكون تلك الأحرف هي بداية السلسلة الحرفية و إلا ستكون النتيجة ٠، أما الدالة strcspn فهي تقوم بتحديد مكان مجموعة من الحروف في سلسلة حرفية، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str1[] = "HHHeeelllooo";
6        char str2[] = "He";
7        char str3[] = "lllooo";
8        int result;
9
10       result = strspn(str1, str2);
11       printf("There are %d character(s) of '%s' in string '%s'\n",\
12             result, str2, str1);
13
14       result = strcspn(str1, str3);
15       printf("First '%s' in string '%s' is start at character %d\n",\
16             str3, str1, result);
17 }
```

البرنامج ٣,٥,٨٦: الدالة strspn و الدالة strcspn

و لكن مرونة الدالتين ليست كبيرة، لذا يفضل أن يتم كتابتهما من جديد على حسب رغبات المبرمج.

٣,٥,١٤,٦ الدالة strpbrk:

تقوم الدالة strpbrk بنسخ سلسلة حرفية و لصقها في سلسلة حرفية أخرى، حيث يجب أن نقوم بتحديـــد بدايـــة النسخة باستخدام أحرف، مثال:

```
1 #include<stdio.h>
2 #include<string.h>
3
4 main(){
5       char str[] = "Hello";
6       char *result = strpbrk(str, "l");
7
8       printf("%s\n", result);
9 }
```

البرنامج ٣,٥,٨٧: الدالة strpbrk

٣,٥,١٤,٧ الدالة strstr:

الدالة strstr مطابقة لدالة strchr في إختلاف بسيط و هي أنها تبحث عن مكان نص بدل حرف واحد، مثال:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  main(){
5        char str1[] = "Hello, World!", str2[] = "World";
6        char *strdest = strstr(str1, str2);
7        int result;
8
9        result = (int)(strdest-str1+1);
10       printf("The word '%s' is at position %d in string '%s'\n",\
11             str2, result, str1);
12
13 }
```

البرنامج ٣,٥,٨٨: الدالة strstr

٣,٥,١٤,٨ الدالة strlen:

تقوم بحسب عدد أحرف سلسلة حرفية، مثال:

```
1 #include<stdio.h>
2 #include<string.h>
3
4 main(){
5       char str[] = "Hello";
6       int result = strlen(str);
7
8       printf("'%s' = %d character(s)\n", str, result);
9 }
```

البرنامج ٣,٥,٨٩: الدالة strlen

٣,٥,١٤,٩ الدالة strerror:

تحمل هذه الدالة مجموعة من رسائل الأخطاء الخاص بالسلاسل الحرفية، مثال:

```
1   #include<stdio.h>
2   #include<string.h>
3   #include<stdlib.h>
4
5   main(){
6       char str[10];
7
8       printf("Enter a string (Max 10 characters): ");
9
10      if((strlen(gets(str)))>10){
11          printf("%s\n", strerror(12));
12          exit(1);
13      }else
14          printf("'%s' = %d character(s)\n",\
15          str, strlen(str));
16
17  }
```

البرنامج ٣,٥,٩٠: الدالة strerror

و يمكن رؤية باقي الرسائل باستخدام التكرار.

٣,٥,١٤,١٠ الدالة strtok:

باستعمال الدالة strtok نحدد مجموعة من الأحرف أو الرموز لسلسلة حرفية حيث لا يتم طباعتها، مثال:

```
1   #include<stdio.h>
2   #include<string.h>
3
4   main(){
5       char *string,
6           str[] = "(Hello, World)",
7           tok[] = " (),I";
8       int i;
9
10      string = strtok(str, tok);
11
12      for(i=0;i<2;i++){
13          printf("%s\n", string);
14          string = strtok(NULL, tok);
15      }
16  }
```

البرنامج ٣,٥,٩١: الدالة strtok

٣,٥,١٥ الملف الرأسي time.h:

يحتوي الملف الرأسي time.h على مجموعة من الدوال الخاصة بمعالجة التاريخ و الوقت، و يحتوي هذا الملف الرأسي على بنية باسم tm و هي معرفة بالشكل التالي:

```
1  struct      tm    {
2          int    tm_sec;
3          int    tm_min;
4          int    tm_hour;
5          int    tm_mday;
6          int    tm_mon;
7          int    tm_year;
8          int    tm_wday;
9          int    tm_yday;
10         int    tm_isdst;
11 };
```

و يحتوي أيضا على متغيرين باسم clock_t و time_t و هما معرفان على الشكل التالي:

```
typedef long time_t;
typedef long clock_t;
```

سنتحدث عن أهم دوال هذا الملف الرأسي، أما الباقي الدوال يمكن رأيتها في الملف الرأسي time.h.

٣,٥,١٥,١ الدالة clock:

تقوم هذه الدالة بالعد، حيث يبدأ عدها عند بداية البرنامج، إبتداءا من الصفر، و قسمة قيمة هذه الدالة على الثابت CLOCKS_PER_SEC أو CLK_PER_SEC يرجع الوقت بالثواني، مثال:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
5          for(;;){
6                  printf("%d\n", clock()/CLOCKS_PER_SEC);
7                  if((clock()/CLOCKS_PER_SEC)==5)
8                          break;
9          }
10 }
```

البرنامج ٣,٥,٩٢: الدالة clock

و يمكن أن نجعل العد بالدقائق أو الساعات أو الأيام، فقط نقوم بقسم الطريقة السابقة على ٦٠ في حالة أردنا العد بالدقائق، و نزيد القسمة على ٦٠ إذا أردنا العد يكون بالساعات، و نزيد أيضا القسمة على ٢٤ إذا أردنا أن يكون العد بالأيام. و باستخدام هذه الدالة يمكن عمل دالة تأخر تساعدنا كثيرا في برامجنا، الدالة ستكون كتالي:

```
1  #include<stdio.h>
2  #include<time.h>
```

```
3
4  void delay(int second);
5
6  main(){
7        int waitsec;
8
9        printf("wait(in seconds): ");
10       scanf("%d", &waitsec);
11       delay(waitsec);
12       printf("Time terminated...\n");
13 }
14
15 void delay(int second){
16       int sec;
17
18       for(;;){
19             sec = clock()/CLOCKS_PER_SEC;
20             if(sec==second)
21                   break;
22       }
23 }
```

البرنامج ٣,٥,٩٣: إنشاء دالة نقوم بالإنتظار لوقت محدد

٣,٥,١٥,٢ الدالة time:

تقوم هذه الدالة بإرجاع عدد الثواني التي مرت من الساعة ٠٠:٠٠ في اليوم ١ جانفي ١٩٧٠، مثال:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
5        time_t Seconds, Minutes, Hours, Days;
6
7        Seconds = time(NULL);
8        Minutes = Seconds/60;
9        Hours   = Minutes/60;
10       Days    = Hours/24;
11
12       printf("%ld\tseconds since 01/01/1970.\n", Seconds);
13       printf("%ld\tminutes since 01/01/1970.\n", Minutes);
14       printf("%ld\t\thours since 01/01/1970.\n", Hours);
15       printf("%ld\t\tdays since 01/01/1970.\n", Days);
16 }
```

البرنامج ٣,٥,٩٤: الدالة time

هنا ستقوم الدالة time بإرجاع قيمة للمتغير Seconds لأن الوسيط الذي تم تمريره إليه فارغ NULL، و يمكن كتابة المثال السابقة بالطريقة التالية:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
```

```
5 	time_t Seconds, Minutes, Hours, Days;
6 
7 	time(&Seconds);
8 	Minutes = Seconds/60;
9 	Hours   = Minutes/60;
10	Days    = Hours/24;
11
12	printf("%ld\tseconds since 01/01/1970.\n", Seconds);
13	printf("%ld\tminutes since 01/01/1970.\n", Minutes);
14	printf("%ld\t\thours since 01/01/1970.\n", Hours);
15	printf("%ld\t\tdays since 01/01/1970.\n", Days);
16 }
```

البرنامج ٣,٥,٩٥: الدالة time (٢)

٣,٥,١٥,٣ الدالة difftime:

لهذه الدالة وسيطين، كلاهما لمتغيرات من نوع time_t، و الدالة تقوم بإرجاع الفرق بين الوقت الأول الموجـــود في الوسيط الأول و بين الوقت الثاني و الموجود في الوسيط الثاني بالثواني،أي تقوم بطرح قيمة الوسيط الأول على الوسيط الثاني، مثال:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
5 	time_t Start, End;
6 	int i;
7
8 	Start = time(NULL);
9 	for(i=0;i<=40000;i++){
10		printf("%d\n", i);
11	}
12	End = time(NULL);
13
14	printf("Loop taken %.0f seconds to terminate...\n",\
15		difftime(End, Start));
16 }
```

البرنامج ٣,٥,٩٦: الدالة difftime

٣,٥,١٥,٤ الدالة localtime:

تقوم هذه الدالة بتحويل عدد الثواني من عام ١٩٠٠ إلى التوقيت محلي ثم تمريرها إلى أعضاء البنية tm، مثال:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
5 	time_t Seconds;
6 	int Year, Month, Day, Hour, Minute, Second;
7 	struct tm* Time;
8
9 	time(&Seconds);
```

```
10
11        Time = localtime(&Seconds);
12        Year = Time->tm_year+1900, Month= Time->tm_mon+1,
13              Day = Time->tm_mday;
14
15        Hour = Time->tm_hour, Minute = Time->tm_min,
16              Second = Time->tm_sec;
17
18        printf("Date: %.4d/%.2d/%.2d\n",Year, Month, Day);
19        printf("Time: %.2d:%.2d:%.2d\n", Hour, Minute, Second);
20 }
```

البرنامج ٣,٥,٩٧: الدالة **localtime**

٣,٥,١٥,٥ الدالة **asctime**:

تقوم هذه الدالة بتحويل بيانات البنية التي تم تمريرها إليها إلى سلسلة حروف من الشكل *DDD MMM D HH :MM :SS YYYY*، مثال:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
5        time_t Seconds;
6        struct tm* Time;
7
8        time(&Seconds);
9
10       Time = localtime(&Seconds);
11
12       printf("Date/Time: %s", asctime(Time));
13 }
```

البرنامج ٣,٥,٩٨: الدالة **asctime**

٣,٥,١٥,٦ الدالة **ctime**:

تقوم هذه الدالة بتحويل عدد الثواني الدالة `time()` إلى سلسلة حروف من الـــشكل *DDD MMM D HH :MM :SS YYYY*، مثال:

```
1  #include<stdio.h>
2  #include<time.h>
3
4  main(){
5        time_t Seconds;
6
7        time(&Seconds);
8
9        printf("Date/Time: %s", ctime(&Seconds));
10 }
```

البرنامج ٣,٥,٩٩: الدالة **ctime**



كل الطرق تؤدي إلى روما، و لنفرض أن روما هي هدفنا في البرمجة (أي البرنامج الذي سنقوم ببرمجته)، و نـضع الطرق التي تؤدي إلى روما هي اللغات البرمجة. سنجد أن الفرق بين طريق و أخرى ليس بكبير، فمثلا ربما نذهب على طريق نجده سهل، أو طريق نجده صعب، أو طريق سريع أو بطيء، أو...، و هذه هو الفرق بين لغة برمجـة و أخـرى، مثال:

الفرق بين لغة C++ و لغة *Assembly* (التجميع) هو:

- ستجد سهولة كبيرة في فهم لغة C++، لأنها لغة عالية المستوى، و من ميزات اللغات العالية المستوى هي جعلها لغة تحاكي لغة الإنسان، أما لغة التجميع هي لغة منخفضة المستوى، و اللغات المنخفضة المستوى هـي لغـات تحاكي لغة الحاسوب مما يجعلها لغة صعبة.
- ستجد حجم برنامج مكتوب بلغة التجميع أصغير من حجم برنامج مكتوب بلغة C++.

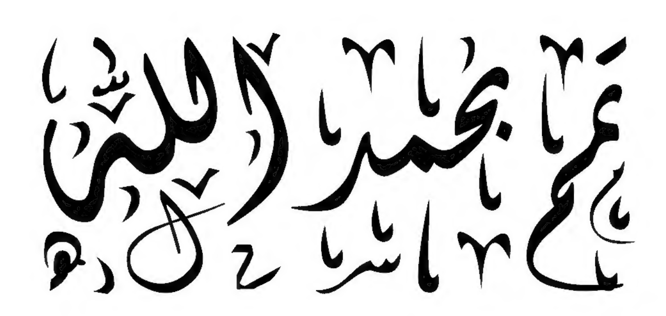
و هذا هو الفرق بين كل لغة و أخرى تقريبا.

خليل أونيس

جدول الأشكال (الصور)

جدول الجداول

جدول البرامج


تم بحمد الله

أهم المراجع

The C Programming Language Book, Second Edition By Brian Kernighan And Dennis Ritchie [Prentice Hall 1988; ISBN 0-131-10362-8]

Fundamentals Programming Family Book, First Edition By IBM International 1985

Turbo C Manuel De Référence Book, By Borland International 1988 [FR]

Wikipedia, the free encyclopedia: www.wikipedia.org

CProgramming.com Your resource for C and C++: www.cprogramming.com

C Pocket Reference Book, By Peter Prinz And Ulla Kirch-Prinz [O'Reilly November 2002; ISBN : 0-596-00436-2]

UNIX System Calls and Subroutines using C Book, By Nikos Drakos 1997

Teach Yourself C in 21 Days Book, By Peter Aitken And Bradley L. Jones [Macmillan Computer Publishing]

The End